1. Abstract

Author:	Mohammad Ayyash
Title of the Thesis:	Quality of Service Analysis of Real Time Applications in Ad Hoc Networks
Date:	10.10.2006 Number of pages: 105
Faculty:	Networking Laboratory Helsinki University of Technology (HUT)
Supervisor:	Professor Raimo Kantola
Instructor:	LSc. Jose M. Costa Requena

Mobile ad hoc network is a research topic that has opened the door for many applications. An ad hoc network is formed between collections of nodes that come together without previous planning, and then they try to optimize their communication to exploit the possibilities of internetworking. One of the applications still under research in this arena is voice communication, or expressed in other words, Voice over IP (VoIP).

VoIP is a voice transmission technology based on packet switched networks. In traditional voice transmission networks, such as PSTN and GSM, circuit switching is used, but in the new generation networks such as 3G and the medium range Wireless LAN technologies, packet switched speech transmission technology is used.

A challenging part of VoIP communication in Ad Hoc networks is to guarantee that the Quality of Service (QoS) does not drop below a certain level. A lot of research is done in trying to give an answer to this challenge, such as using a robust routing protocol, or introducing more efficient voice codecs, or incorporating two physical channels, one for data and one dedicated for real-time and voice communications.

This report presents experiments to measure the quality factors of an audio session in a WLAN Ad Hoc network. For such a purpose, a VoIP software client is designed and implemented, and a set of test cases are designed and executed to collect metrics related to QoS factors. Finally, analysis of collected data is performed and conclusions are drawn.

Keywords: Voice over IP, VoIP, Ad Hoc, SIP, RTP, GSM

Kirjoittaja:	Mohammad Ayyash
Otsikko:	Reaaliaikasovellusten QoS-analyysi Adhoc-verkoissa
Päivämäärä:	10.10.2006 Number of pages: 105
Osasto:	Tietoverkkolaboratorio Teknillinen Korkeakoulu (TKK)
Valvoja:	Professori Raimo Kantola
Ohjaaja:	LSc. Jose M. Costa Requena

Mobiili Adhoc-verkko on tutkimus aihe, joka on avannut oven monille sovelluksille. Se muodostuu eri solmuista jotka liittyvät toisiinsa ilman mitään suunnitelmaa, yrittäen hyödyntää tehokkaasti verkottumisen mahdollisuuksia. Yksi tutkimuksen alla olevista sovelluksista on puheliikenne, toisin sanottuna Voice over IP(VoIP).

VoIP on tiedonsiirtoteknologia, joka perustuu pakettiverkkoihin. Perinteisissä puheliikenneverkoissa, kuten PSTN ja GSM, käytetään piirikytkentää, mutta uuden sukupolven verkoissa, kuten 3G ja WLAN käytetään pakettivälitteistä puheensiirtoteknologiaa.

Haastavinta VoIP kommunikaatiossa tilapäisverkoissa on taata, ettei palvelunlaatu (QoS) putoa tietyn tason alapuolelle. On tehty paljon tutkimusta yrittäen antaa vastausta tähän haasteeseen, kuten tehokkaan välitysprotokollan käyttäminen, tehokkaammat äänikoodekit tai kahden fyysisen kanavan yhdistäminen; toinen datalle ja toinen reaaliaikaiseen puheviestintään.

Tämä raportti esittelee kokeilun audioyhteyden laatutekijöiden mittaukseen väliaikaisessa langattomassa verkossa. Kokeeseen suunniteltiin ja toteutettiin VoIP asiakasohjelma, ja joukko suunniteltuja testitapauksia suoritettiin palvelunlaatuvaatimustekijöihin liittyen. Lopuksi kerätty data analysoidaan johtopäätöksineen.

Avainsanat: Voice over IP, VoIP, Ad Hoc, SIP, RTP, GSM

2. Preface

This master thesis has been written at the Networking Laboratory of Helsinki University of Technology.

Date

10.10.2006

Helsinki, Finland

Author: Mohammad Ayyash

3. Contents

4. Abbreviations

AMR	Adaptive Multi-Rate Codec
AMR-WB	Adaptive Multi-Rate Codec Wideband
AP	Access Point
CSRC	Contributing Source
DSP	Digital Signal Processing
ETSI	European Telecommunications Standards Institute
FIFO	First-In-First-Out
GSM	Global System for Mobile Communications
HTTP	Hypertext Transport Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISO	International Standards Organization
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
Kbps	Kilobits per second
LAN	Local Area Network
MAC	Medium Access layer
Mbps	Megabits per second
PCM	Pulse Code Modulation
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RTCP	Real-Time Transport Control Protocol
RTP	Real-Time Transport Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
TCP	Transport Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
WLAN	Wireless Local Area Network
VoIP	Voice over Internet Protocol

5. Figures

6. Tables

Chapter 1

1. Introduction

Voice over IP (VoIP) is an active research area. VoIP applications can facilitate group work, such as a building site, or a rescue operation, such as firefighting. It is the nature of ad hoc networks in a WLAN environment that there is no fixed topology and there are no guarantees of a permanent link. This environment will affect both speech conversation and people's reaction to such technology.

1.1. Research problem

The goal of this research is to perform real time audio conversations over ad hoc WLAN networks and record the Quality of Service (QoS) factors under a set of conditions. In this research, different kinds of scenarios will be applied.

The design and implementation of a VoIP software client is a part of this research, too. This research will tackle the obstacles found in real-time applications, such as, voice encoding and decoding, processing power, and the Quality of Service requirements.

Two routing protocols will be used, Optimized Link State Routing Protocol (OLSR) and Ad-hoc On-demand Distance Vector (AODV). The reason for this selection is that they come from two different families of protocols, proactive routing (OLSR) and reactive routing (AODV). OLSR and AODV satisfy some but not all the requirements of Quality of Service. One of this thesis objectives is to compare the performance of OSLR and AODV during audio calls in ad hoc networks.

1.2. Tasks and steps

In this thesis, a VoIP software client is designed and implemented. The VoIP client includes the SIP signaling protocol, the RTP media transport protocol, the GSM voice codec, and real-time audio management.

After the VoIP client software implementation is completed, a set of test cases is designed and executed. The test cases are used to verify:

- The effect of increasing the number of hops between the two calling ends.
- The effect of changing the type of intermediate routing nodes.
- The effect of network topology changes during a voice call.

All test cases will be performed using both OLSR and AODV routing protocols to draw a comparison between the two routing protocols.

1.3. Structure of the thesis

Chapter 2 introduces VoIP protocols; mainly, the SIP signaling protocol and the RTP media transport protocol. This chapter also presents audio operations such as sampling and real time audio management.

Chapter 3 presents the factors affecting Quality of Service. These factors will be measured during the testing process. Chapter 4 presents the design and implementation of the VoIP software client. A VoIP software client is designed for this research to enable the of fine tuning test case parameters. Chapter 5 presents the execution of test cases. Chapter 6 presents the conclusions. Chapter 7 includes suggestions, ideas, and future work.

Chapter 2

2. Introduction to Voice over IP

This chapter will describe the protocols used in VoIP. It will also present basic audio fundamentals and real time audio management, which are necessary to understand VoIP operations. Finally, this chapter will go through the affecting quality of service factors.

2.1. Voice over IP protocols

VoIP protocols are divided into two major areas; signaling and media transport. The session initiation protocol (SIP) [4] is used for signaling and the Real Time Transport Protocol (RTP) [5] is used for media transport. Both SIP and RTP are application layer protocols. Their relation to OSI model and the TCP/IP stack is illustrated in Figure 1

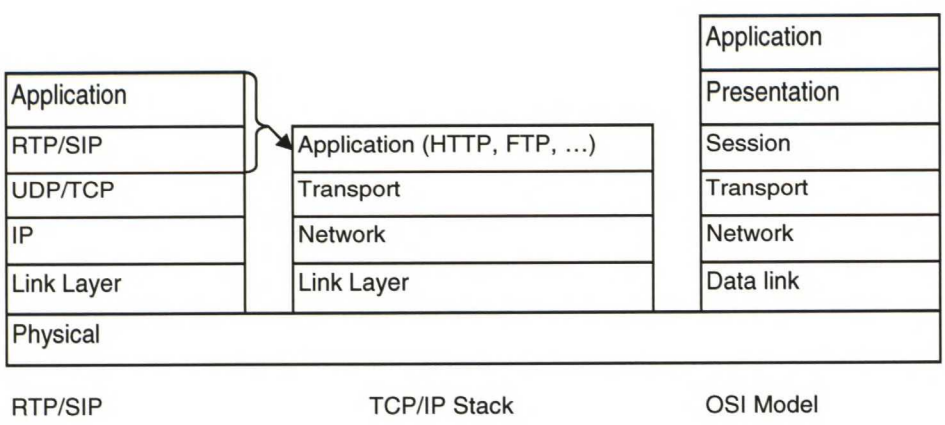


Figure 1 Protocol Stack

2.1.1. Session Initiation Protocol

Signaling establishes an agreement on audio parameters between the two calling parties before any audio conversation can take place, not to mention that one end must learn about the other in order to contact it. The Session Initiation Protocol (SIP) is one of the signaling protocols. There are other signaling protocols in the realm of packet switched voice communications, such as the ITU-T H.323 protocol [6].

SIP was developed by IETF in RFC 3261 [4]. It is a simple, scalable and flexible protocol because it is similar to HTTP. However, it is inefficient because a SIP agent must work its way through textual headers and be prepared to handle different potential header values. A SIP header name must be present followed by its value, unlike other protocols like IP, where only the value must be present in a specified location in the header of the packet. RFC 3261 [4] extensively covers the usage of SIP header names.

SIP is also used to establish sessions of different kinds, for example:

- Establishing an internet telephone call.
- Distribution of multimedia.
- Multimedia conferences.
- Distributed computer games.

A session refers to the participants of the target collaboration, such as audio conversation, in addition to the state maintained by each participant regarding the ongoing collaboration. SIP protocol provides a common communication language to set up, maintain and tear down the collaboration. SIP does not provide the means of communications or state management. This is provided by lower layers protocols and participants software logic.

In this SIP paradigm, the logic and state management is shifted towards the terminals. This is unlike PSTN networks, which concentrate the logic and state management in the network and leaving terminals dummy. The SIP protocol design leaves a large room for implementing a wide variety of services without being limited by the network capabilities, or having to change the network. The network is there only to facilitate communications between terminals. [7]

SIP Architecture

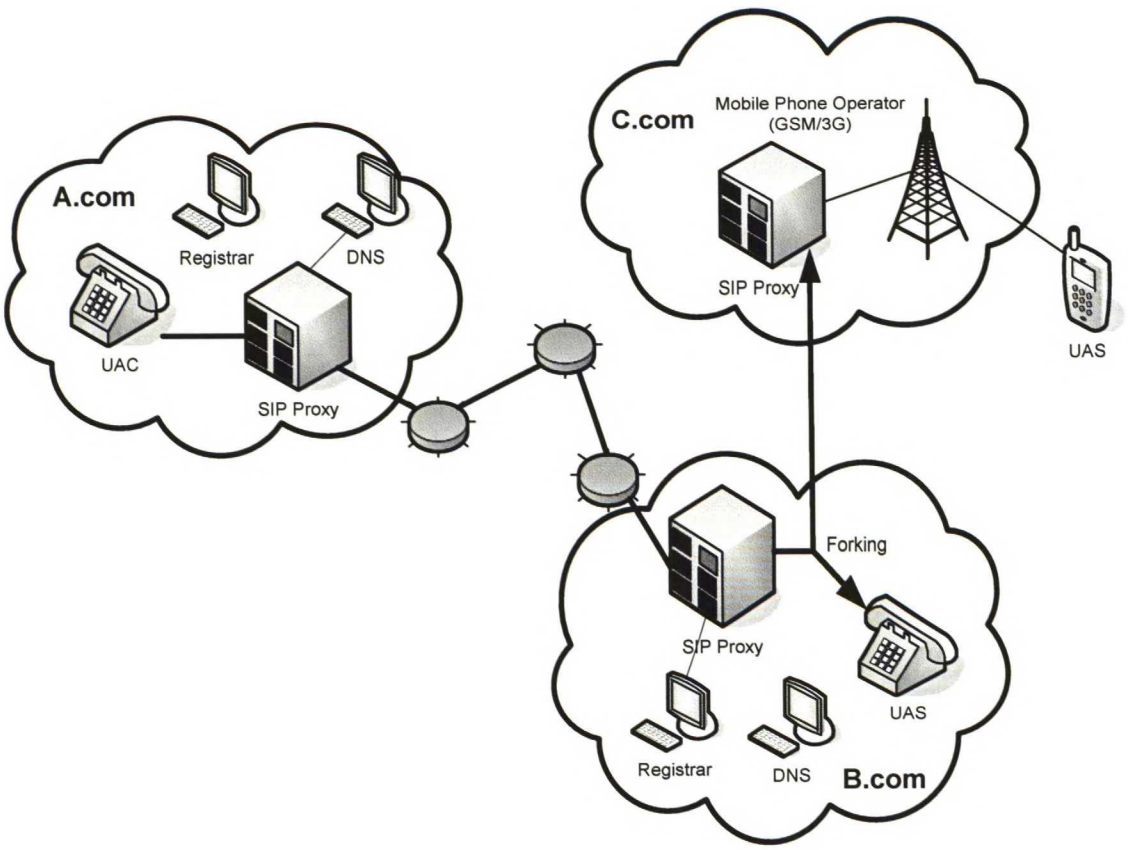


Figure 2 SIP Network

Figure 2 illustrates a general SIP infrastructure example. There are three operators, A.com, B.com and C.com. User A in A.com tries to call a user B in B.com domain. The SIP enabled phone of user A issues a SIP INVITE message addressed to user B@B.com. The SIP proxy in A.com domain receives the request and tries to find B.com domain by consulting the DNS. The DNS receives the request from the SIP proxy of user A and returns the address of the SIP proxy of user B. The SIP proxy of user A directs the INVITE request to user B proxy. The SIP proxy in B.com consults the SIP Registrar to learn about the current location of user B. The proxy of user B then forwards the INVITE message to the land phone of user B and sends back 180 RINGING response to the SIP proxy in A.com domain. The SIP Proxy in B.com domain is state-full and therefore is able to intelligently fork the call. User B has programmed the SIP proxy in B.com to redirect the call to his mobile phone in case the call is not picked up from the

land phone. Then, after some time out, if the phone call is not picked up on the land phone, the SIP proxy of B.com redirects the call to the SIP proxy in C.com. The SIP proxy of C.com is part of the mobile phone operator network and interfaces with the rest of the mobile network. It forwards the call to the mobile phone of user B. When user B picks up the call, a 200 OK message is sent back to user A. Now, user A and user B can directly reach each other, and continue the set up of the audio/video conversation.

Setting up the multimedia session in the above scenario is facilitated by including a description in the SIP messages bodies. This description specifies the intended session type and the available capabilities of both user A and user B. A standard protocol for such description is the Session Description Protocol (SDP).

User B can also issue a REGISTER request to his home SIP registrar from his SIP enabled mobile phone. The registrar will inform the SIP proxy of B.com to immediately redirect the phone call to C.com mobile operator. When user B returns to his office, the mobile phone, through location and presence services, can issue another REGISTER request and instruct the registrar to inform the SIP proxy in B.com to direct the calls to the land phone as usual.

The above scenario clearly shows the flexibility of SIP infrastructure. Implementing the logic mentioned in the scenario only needs programming the SIP proxy in B.com. Figure 3 shows the flow diagram of this scenario.

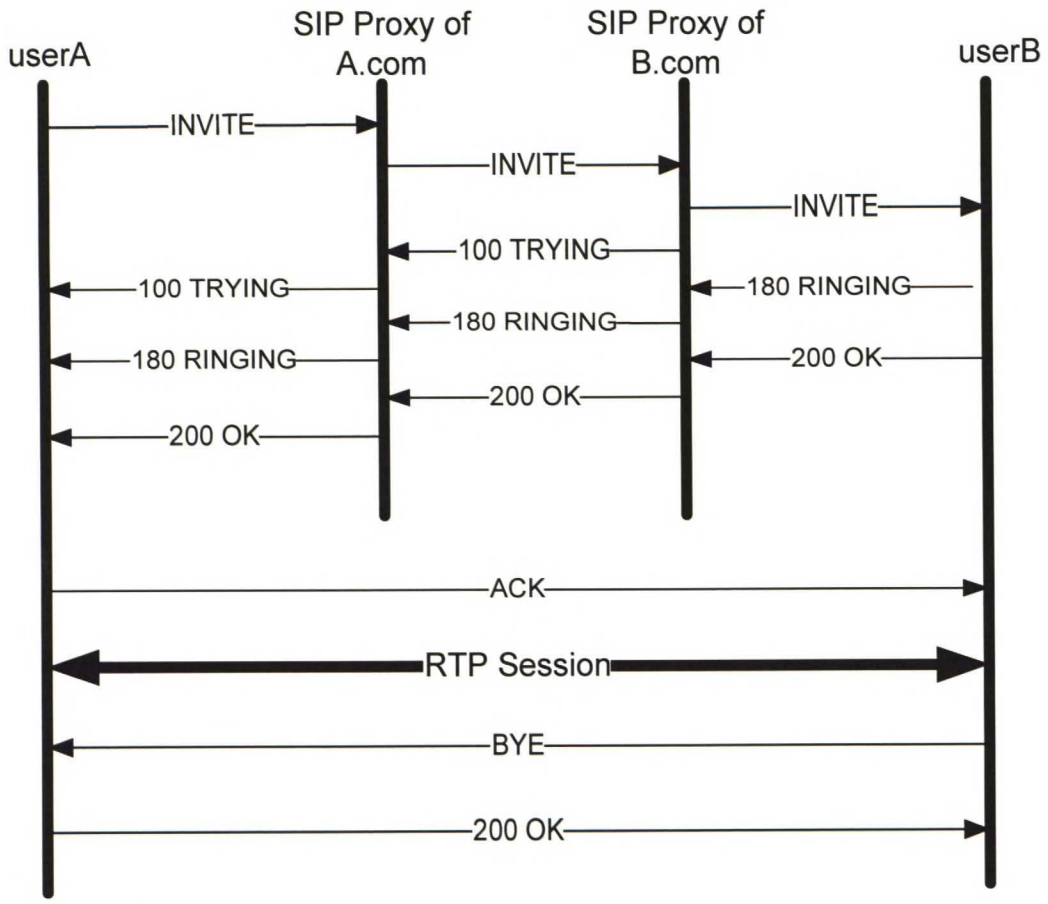


Figure 3 SIP Flow Diagram

2.1.2. Session Description Protocol

The Session Description Protocol (SDP) is an application level protocol. SDP describes multimedia sessions for the purpose of session announcement and initiation. SDP is a short structured textual description of the session. SDP is intended to be a general purpose protocol and can be used for a wide range of applications.

SDP is defined by IETF in RFC 2327 [8]. SDP messages are text messages which consist of fields. SDP messages are exchanged in the body of SIP messages. Table 1 presents SDP fields.

Table 1 SDP fields

Char	Meaning
v	protocol version
o	owner/creator and session identifier

s	session name
i	session information (Optional)
u	URI of description (Optional)
e	email address (Optional)
p	phone number (Optional)
c	connection information - not required if included in all media (Optional)
b	bandwidth information (Optional)
One or more time descriptions	
t	time the session is active
z	time zone adjustments (Optional)
k	encryption key (Optional)
a	zero or more session attribute lines (Optional)
r	zero or more repeat times (Optional)
Zero or more media descriptions	
m	media name and transport address
i	media title (Optional)
c	connection information - optional if included at session-level (Optional)
b	bandwidth information (Optional)
k	encryption key (Optional)
a	zero or more media attribute lines (Optional)

2.1.3. Real time Transport Protocol

Real time transport protocol (RTP) is used to exchange real-time multimedia data, including audio, video, and text, between end points. RTP protocol facilitates splitting, packing, sending and receiving real time data over the Internet.

Mixers and Translators

RTP supports low speed links using RTP mixers; it includes security elements such as RTP translators. A mixer will combine all audio sources into one stream. It will then change the used compression to a lower bandwidth codec suitable for a low speed link. Figure 4 shows an RTP mixer scenario.

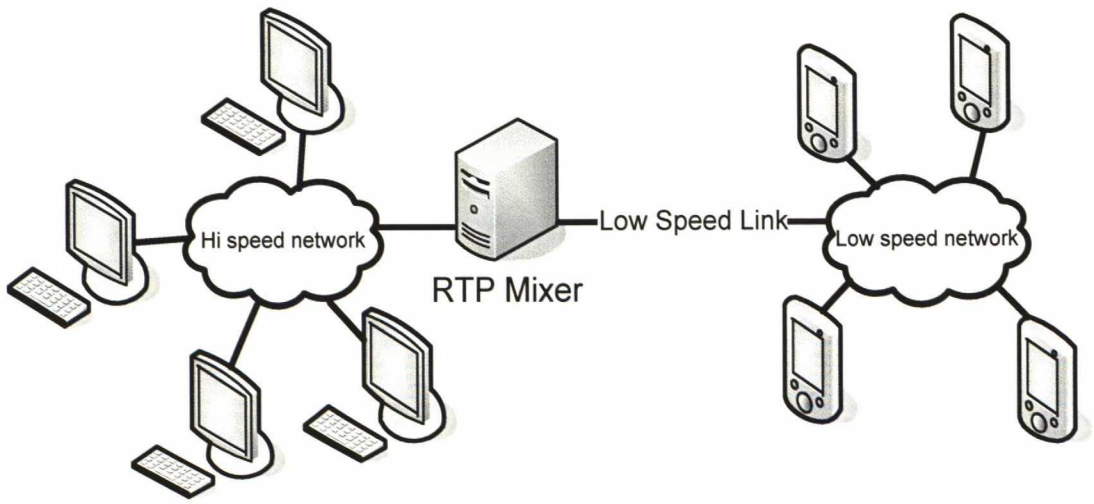


Figure 4 RTP network

If some end points are located in a private secured network behind a firewall, RTP translators can be used on both sides of the firewall to establish a secure tunnel through the firewall. This arrangement allows joining a media conference between end points across a firewall. Figure 5 illustrates a scenario including RTP translators.

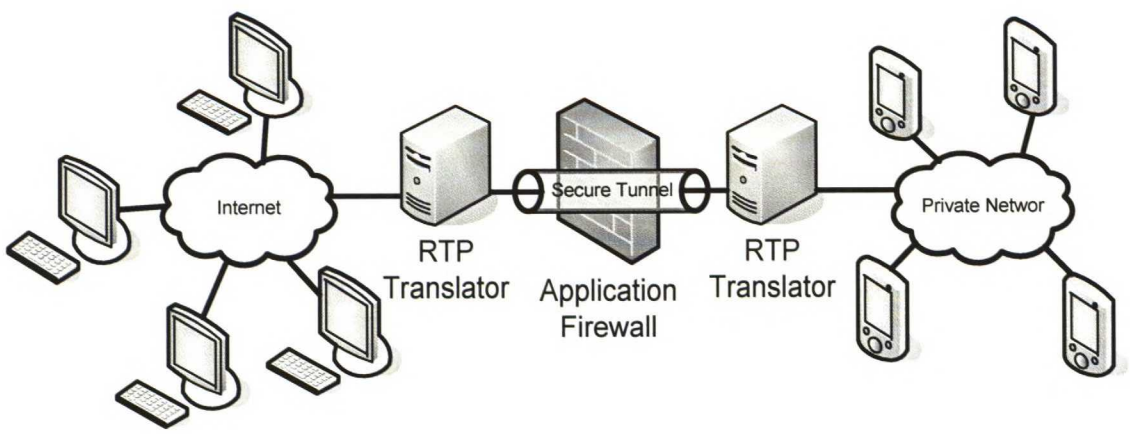


Figure 5 RTP network with RTP translators

Figure 6 shows one RTP packet sent by the VoIP software client developed for this project. Figure 6 also details the RTP header. The payload of the RTP packet shown in Figure 6 is three GSM audio frames.

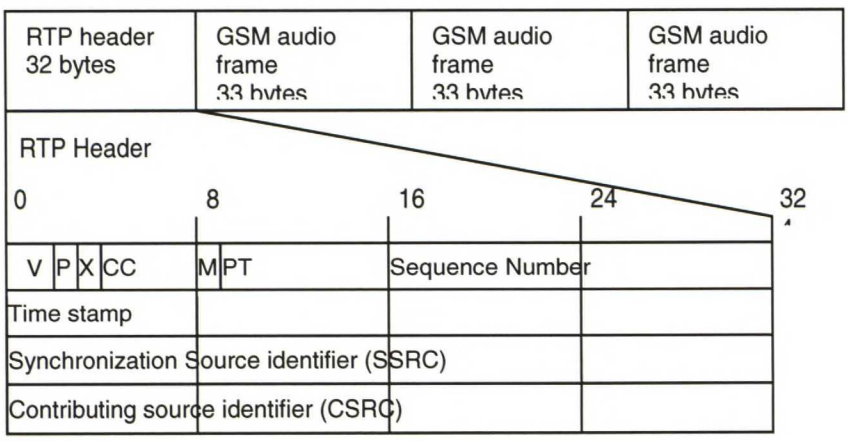


Figure 6 RTP header fields

2.1.4. Routing Protocols

The challenge of ad hoc networks for routing protocols is the dynamic nature of ad hoc networks, which includes a constantly changing topology, and the absence of an infrastructure. Each node must discover the network topology and keep track of changes in the topology. The basic requirements of a successful routing protocol in ad hoc network are [9]:

- Small routing table. This is an important consideration in nodes with small memories or reduced processing capacity.
- Ability to select the best route based on different selection criteria: fastest, highest throughput, most reliable or least number of hops. As an example, a routing protocol can select routes based on the lowest transmission power required, by taking into account the battery life time when calculating the next hop. Example of power aware protocols are: ISAIAH, PARO, EADSR and PAMAS [10]
- Low number of exchanged messages.
- Agility in updating topology information. Topology of the network can change very often in ad hoc networks. The protocol must be able to react in an optimum way.

Routing protocols can be classified according to the employed routing algorithms; link state algorithms and distance vector algorithms.

Ad hoc On Demand Distance Vector (AODV)

Ad hoc On Demand Distance Vector (AODV) is an example of distance vector routing. In distance vector routing, every node starts up by discovering neighboring nodes and constructing a vector to all other nodes, including the cost to each node. Then each node regularly exchanges the routing vector with all neighboring nodes. Initially, a node will know the distance to its immediate neighbors. Over time, every node will have a complete route information to all other nodes in the network and the cost of each route. If a node is down, its immediate neighbors will detect that and update their routing vectors. Then, the immediate neighbors will exchange this information with their respective neighbors until all the nodes in the network update their routing tables so as not to include the failing node.

On demand distance vector routing is a reactive routing protocol. It looks up a route only when needed. When a node needs a connection to another, it broadcast a route request message in the network. Other nodes reply back if they know how to reach the destination node, they also save the information about the requesting node. The requesting node will then calculate the best route from the replies it received.

The advantages of AODV are that it does not add traffic overhead once the route has been discovered, it is simple and it does not involve complex computation. On the other hand, discovering the route to the destination node can be slow. [12]

Optimized Link State Routing (OLSR)

In a link state routing protocol a node builds a map of links to neighboring nodes, and it floods this information to the entire network. Each node builds a map of the network topology based on the link updates it receives from other nodes. Dijkstra's algorithm of calculating the routes is usually utilized in such protocols.

The advantages of optimized link state routing protocols are that they stabilize quickly, they do not generate excess traffic, and they respond fast to topology changes. However, the disadvantage is that they need to store a large amount of data in each node [13].

The Optimized link state routing protocol is proactive. A node continuously tries to discover the network even if there are no topology changes. The advantage is faster link establishment

between two nodes, since information about the network topology is up to date in every node. On the other hand, this means that the nodes have to continuously send and receive routing information, which creates excessive traffic overhead, uses more bandwidth, and requires more processing and larger memory storage [14].

Comparison between OLSR and AODV

AODV is suitable in ad hoc networks when the applications are tolerant to the high cost of initial link establishment, and when they expect to communicate with the same destination for long periods of time. For example, file sharing application will benefit from AODV protocol. OLSR is faster in establishing a link. OLSR is suited for applications that need fast establishment of connections and are not affected by routing protocol higher share of bandwidth. For example, chatting applications will benefit from this OLSR protocol.

Real time applications, e.g. VoIP, are tolerant to latency in establishing the initial link. This can be counted against the time spent during signaling to establish the call. Nevertheless, a fast link re-establishment is required if the topology changes during the call. During media transportation, the bandwidth is precious and a protocol with lower share of bandwidth usage is needed. It follows that OLSR and AODV satisfy some but not all the requirements of real time applications, and a comparison between OSLR and AODV performance under audio calls in ad hoc networks is one objective in this thesis.

2.1.5. Wireless and Ad Hoc Networks

There are many types of wireless technologies designed for communications, such as Bluetooth, IrDA, WiFi, WiMax. Wireless LAN (WLAN) is the local area network composed of the nodes communicating with each other using a wireless access technology. 802.11 and its variants specifications are the most popular medium access protocols for WLAN networks. WLAN has two modes of operations: managed and ad hoc modes. A wireless network is in the managed mode when an infrastructure is used. Fixed access points are used to connect the wireless network to wireline backbones. The access points manage the network by providing services such as DHCP, routing and gateways.

In ad hoc mode, wireless nodes communicate directly with each other, additionally, there is no infrastructure to provide any kind of services or centralized management. The nodes in such a network have to organize themselves to provide common services. For example, intermediate

nodes will act as routers between other nodes and each node will advertise the services it implements such as printing, access to wired backbone (gateway service), or voice communication.

Another feature of ad hoc network is the continuously changing topology. In ad hoc networks, some new nodes appear, existing nodes disappear, new routes are discovered, and old routes become unavailable. Nodes should be prepared to react to topology changes.

In this project, WLAN network in ad hoc mode is used. The medium access protocol is 802.11b. The nodes used to create the network are either iPAQs or laptops. All are equipped with WLAN network interface cards.

2.1.6. SIP In Ad Hoc networks

This thesis focuses on the usage of SIP in an end to end fashion without the existence of an infrastructure. The SIP services implemented in this project are messaging and call setup only.

The SIP user agents are application layer elements, they are not aware of the ad hoc network details, such as the topology or routing protocol. Figure 7 illustrates the topology used in this project.

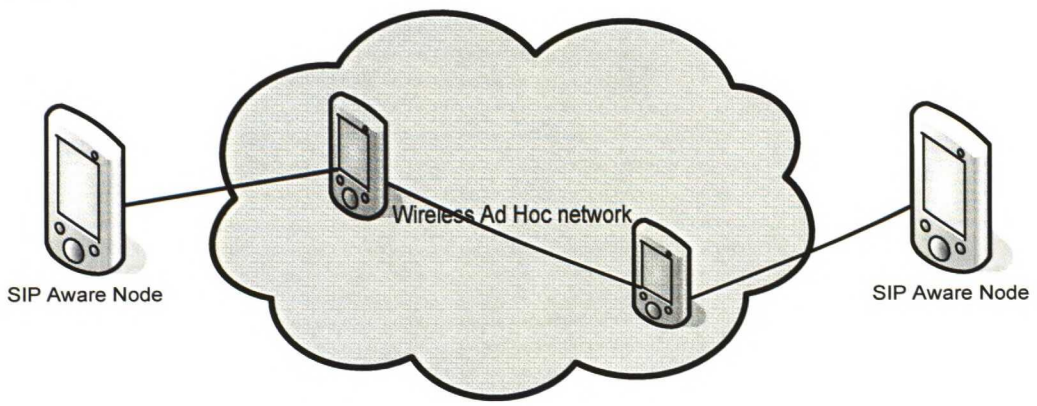


Figure 7 SIP Network in this project

Figure 8 illustrates the flow diagram of the call setup and messaging scenarios used in this project.

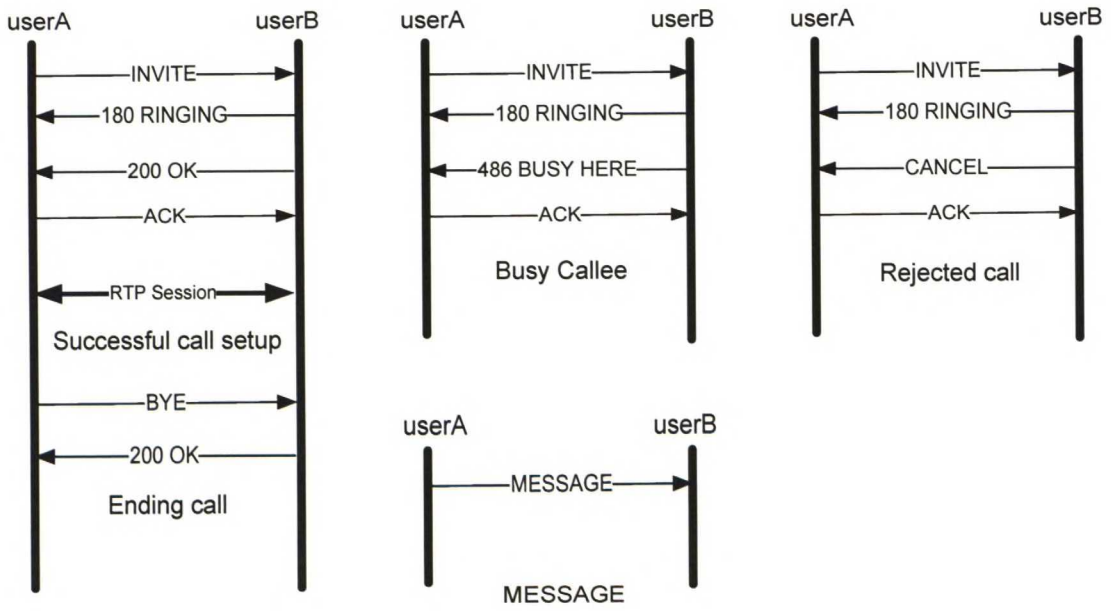


Figure 8 SIP Signaling implemented in this project

Because there is no infrastructure in ad hoc networks, a SIP proxy or SIP registrar cannot be used during the call setup phase. Additionally, since there is no DNS, it is also not possible to address a callee by the email style address i.e. userB@B.com. Thus, in ad hoc networks, a callee is can addressable by his IP address.

2.2. Audio Fundamentals

Figure 9 illustrates the process of recording audio on the sender part and playing back audio at the receiver part. These operations are the focus of this section.

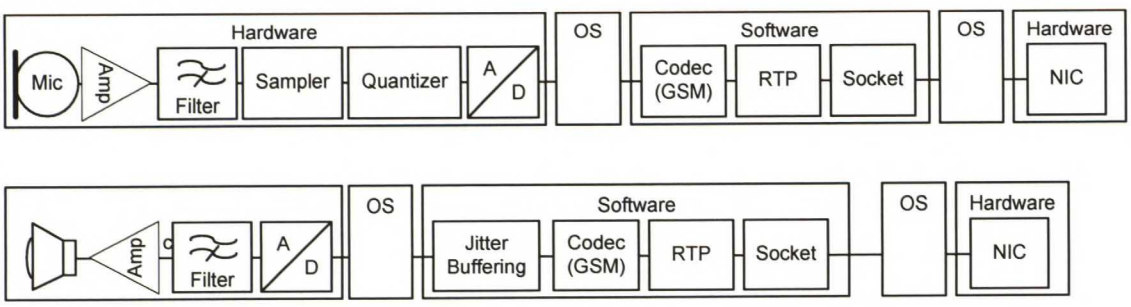


Figure 9 Basic audio operations

2.2.1. Voice Coding Principle

Audio has to be digitized in order to transmit it over data networks. Audio digitization starts by sampling analog audio signal at a sampling frequency. According to Nyquist theorem, at least double the maximum frequency of the sampled signal must be used in sampling in order to maintain the original signal shape. A lower sampling frequency will result in aliasing. Aliasing will distort the frequency response of the signal. In practice, a slightly higher frequency than the maximum frequency of the audio signal is used for sampling, because the electronic components used in constructing frequency filters are not ideal. Figure 10 illustrates sampling in frequency domain. It also shows the difference between ideal filters and practical filter.

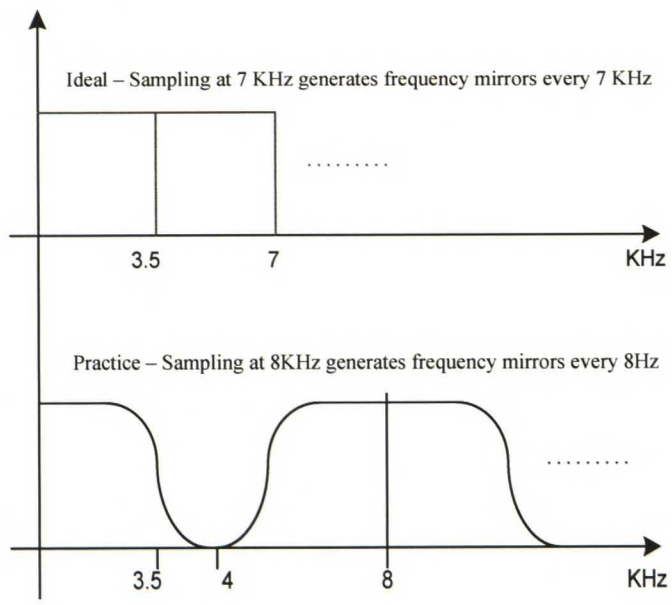


Figure 10 Sampling frequency domain

Given that the human conversation voice is in the range of 50 Hz to 3.5 KHz, sampling frequency should be more than 7 KHz. In practice a higher sampling frequency is used. Usually this frequency is 8 KHz.

Figure 11 shows the time domain response of the sampled signal. The analog values sampled are then quantized, and digitized into bits.

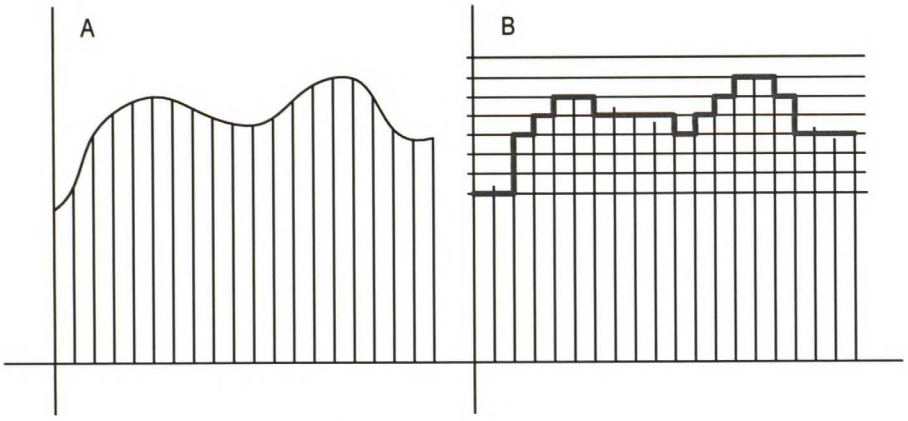


Figure 11 Sampling in time domain

Figure 11-A shows the analog signal and Figure 11-B shows the quantization result of the sampling output. The vertical lines represent the samples instances. They are spaced by 1/f, where f is the sampling frequency. The analog samples values are then input to a quantization circuit that rounds the values to predefined levels. The number of levels depends on the resolution of the A/D converter. If the A/D converter is 8 bits wide, the number of levels will be $2^8 = 256$ levels. In this case, for each analog value, the output is an 8 bit digital value.

Figure 11-B also shows that there are quantization errors. Those analog input values that do not lie exactly on a level will be rounded up or down to the nearest level. For this reason, the original signal shape will not be maintained. The higher resolution of the A/D, the more levels it can supports, and the more accurate it can maintain the original analog signal shape.

2.2.2. Audio Codecs

Codecs are data compression techniques. Codecs are used to reduce the size of the digital data produced from A/D conversion. Codecs are needed when transmitting over a narrow bandwidth channel. A good codec can take advantage of audio properties, for example, a property of human voice is that it falls in the frequency range below 3.5 kHz.

Bandwidth usage and audio quality are two contradicting factors. Using a high resolution A/D in sampling will produce more accurate digital representation of the analog audio signal, but will also produce a larger amount of digital data. Codecs try to balance between the bandwidth usage and audio quality. There are several kinds of codecs, Table 2 list some voice codecs and their properties [15].

Table 2 Voice Codecs used in VoIP

Codec	Sampling rate (kHz)	Bit rate (kbps)	delay frame + look ahead (ms)
Speex	8, 16, 32	2.15-24.6 (NB) 4-44.2 (WB)	20+10 (NB) 20+14 (WB)
iLBC	8	15.2 or 13.3	20+5 or 30+10
AMR-NB	8	4.75-12.2	20+5
AMR-WB (G.722.2)	16	6.6-23.85	20+5
G.729	8	8	10+5
GSM-FR	8	13	20+
GSM-EFR	8	12.2	20+
G.723.1	8	5.3 6.3	37.5
G.728	8	16	0.625
G.722	16	48 56 64	20

In this project, GSM codec is selected because it is freely available, and it has an open source implementation for Linux [17], additionally, it has low processing requirements and it does not need buffer look-ahead to encode audio (look ahead will add to the delay budget). The GSM full rate speech codec operates at 13 Kbits/s and requires an audio input stream sampled at 8000 Hz and 13 bits per sample.

The GSM codec encodes 20 msec worth of audio frame in one GSM audio frame. The uncompressed GSM audio frame length is:

20 (msec) * 8 (sample/msec) * 13 (bits/sample) / 8 (bit/byte) = 260 bytes.

The GSM codec generates 33 byte of compressed audio, with a compression ratio 33/260 = 12.69%

The input audio stream rate is 260 bytes every 20 msec, yielding a 104 kbps data stream, the output audio stream rate is 33 bytes every 20 msec, yielding a 13.2 kbps data stream. Figure 12 summarizes GSM codec compression operation.

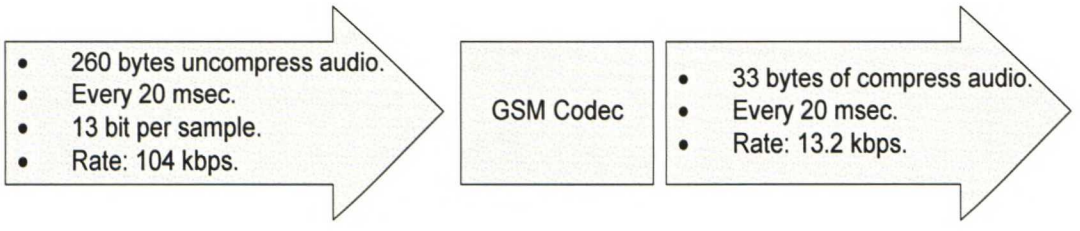


Figure 12 GSM codec operation

In this project, the open source GSM codec implementation [17] uses 16 bits per sample to match the sound card output of either 8 bits or 16 bits. The GSM codec simply ignores the 3 least significant bits.

2.3. Real Time Audio management

2.3.1. Delay Budget

The delay budget is the amount of delay allowed between the time of recording audio at the sender part and the time of playing it back at the receiver part. ITU-T standard G.114 [3] recommends that the one-way delay should be kept below 150 ms for an acceptable conversation quality. A delay higher than 400 msec is unacceptable.

The delay budget is broken up into the following items:

- Recording time: it is the time required by the audio device hardware to record audio and to place it in a buffer.
- Sender part processing: it is the time that includes compression, packing into RTP packets and other logic performed by the VoIP software client.
- Network delay: it is the time that includes packet transmission and routing in intermediate nodes.
- Receiver part buffering: it is the time that the packet is placed in the jitter buffer.
- Receiver part processing: it is the time that includes unpacking and decoding the received RTP packets, and other logic performed by the VoIP software client.

Figure 13 shows these operations. The delay needed to playback the audio frame is not included in the budget. This is indicated in Figure 13 by the end to end delay.

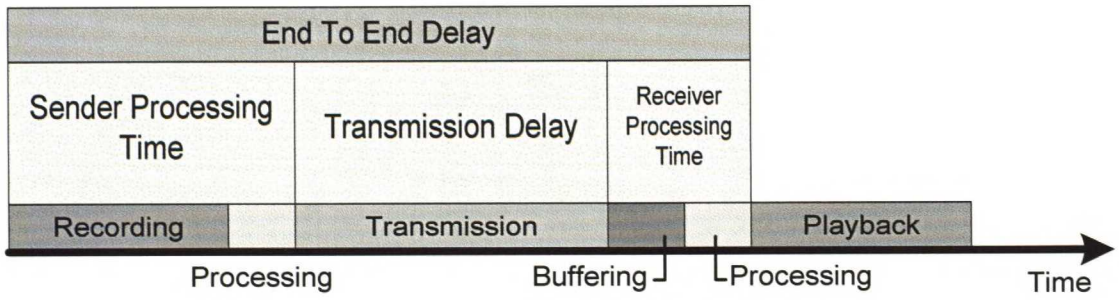


Figure 13 Delay budget

Table 3 shows typical values of the delay budget.

Table 3 Delay budget

Operation	Delay (msec)
Recording	60
Sender part Processing	5-6
Network delay (1 hop, no routing)	~15
Receiver part buffering	~60
Receiver part processing	5-6
Total	147

The delay budget becomes more complex when more nodes are involved in routing between the two end nodes. The network delay increases to the point that it might go over the acceptable limit of 400 msec. This is a limiting factor of the number of intermediate routing nodes for an acceptable audio quality in an ad hoc network.

2.3.2. Realtime Analysis

Real time audio management can be analyzed by plotting RTP packets versus time. Figure 14 shows a typical plot of realtime analysis. The sequence number of RTP packets is used on the Y-axis, and time is used on the X-axis.

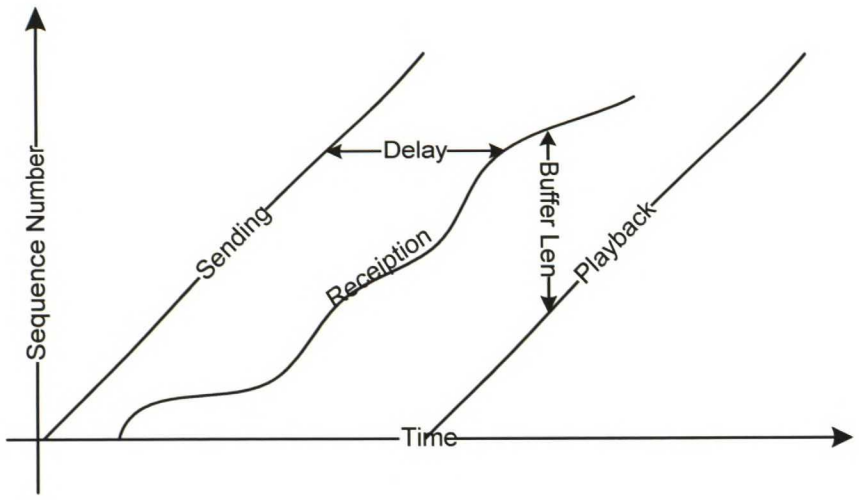


Figure 14 Timing diagram

The sending line shown in Figure 14 is a straight line. It means that the sequence number is increasing steadily with time, representing a fixed rate transmission. The slope of sending line has the unit of packets per second, which is directly related to transmission bit rate (byte per second). Higher bit rates result in a steeper sending line.

The playback line represents the RTP packets playback instances. This line should be parallel to the sending line. If one uncompressed audio frame is recorded every 20 msec, the playback of one uncompressed audio frame should happen every 20 msec.

The reception curve represents the time when RTP packets are actually received at the receiver. It is not a straight line because packets arrive after a varying delay.

The horizontal distance between the sending line and the receiving curve represents the network delay. The vertical distance between the receiving curve and the playback line represents the buffered amount of packets.

Figure 15 presents a scenario to show how the receiver manages packet buffering and playing back. This scenario shows how the receiver part copes with late packet arrivals attributed to increased network delay.

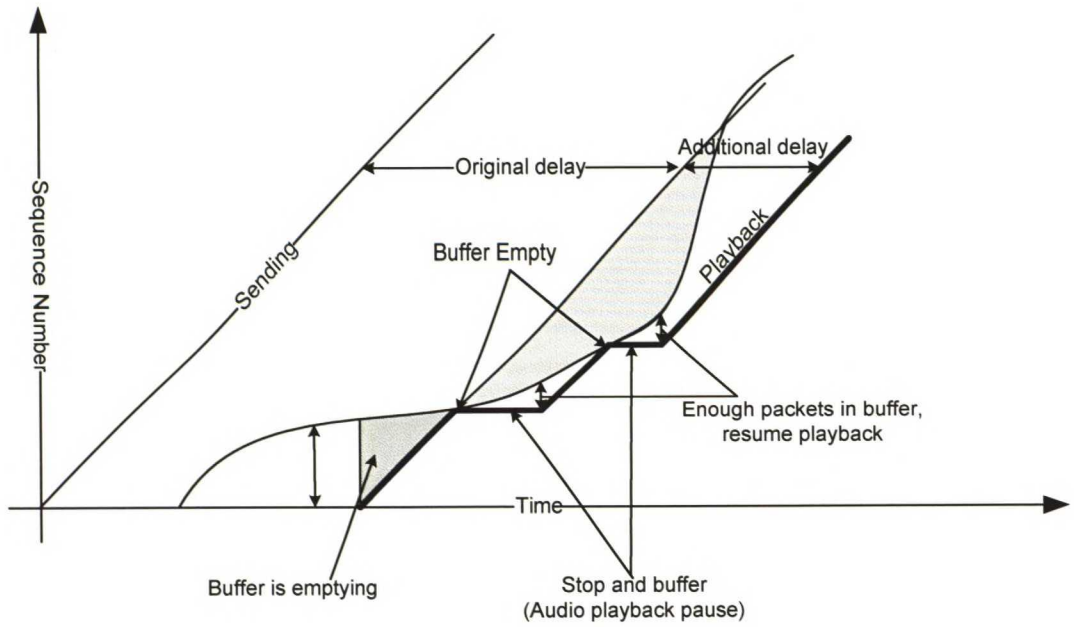


Figure 15 Timing diagram with buffer under run

When the receiver has buffered enough packets in the jitter buffer, it starts playing back and the playback line starts from the time axis as shown in Figure 15. The scenario shows that the network delay is continuously increasing and the jitter buffer content is progressively being depleted. Eventually, the jitter buffer will become empty. This event is refereed to as buffer under run. The receiver will stop playing back and begin buffering arriving packets to build a new jitter buffer; and the audio is stopped during this time. Once the jitter buffer contains enough packets, the receive starts draining the jitter buffer again, and audio is resumed. It is important to understand that after each stop to buffer, the total end to end delay increases.

One of the problems facing the receiver part is how to keep the increasing end to end delay under a certain level. Every time a buffer under run occurs, the end to end delay increases. In the scenario shown in Figure 15, the end to end delay increased too much. The receiver has to go back to the original playback line in order to reduce the end to end delay, as shown in Figure 16

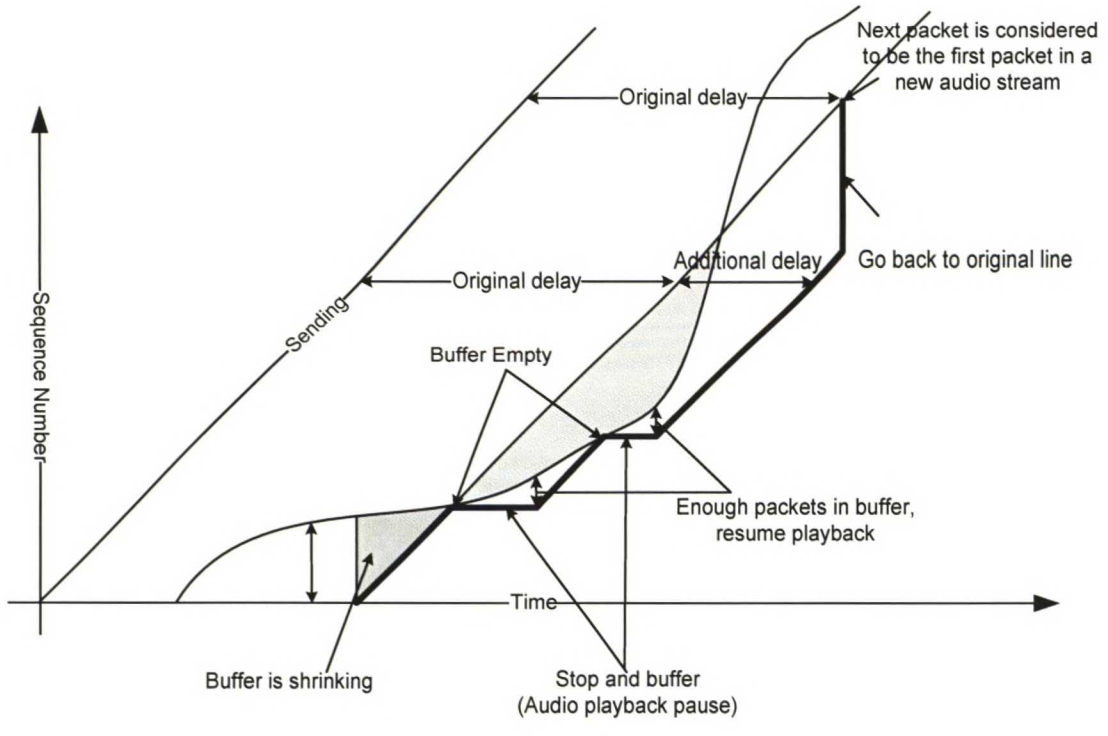


Figure 16 Timing diagram and packet flushing

The vertical section of the playback line shown in Figure 16 represents going back to the original playback line. This vertical line can be achieved by skipping the packets in the jitter buffer and assuming that the next received packet is the start of the audio conversation. Therefore, the vertical line represents flushing the jitter buffer.

Buffer flushing has a negative impact on audio quality. Buffer flushing will be perceived as a skipped section in the audio. The receiver part has to balance between how often to flush the jitter buffer, and the current size of the jitter buffer. As a design rule in the VoIP software client, the receiver part is not allowed to flush the buffer too often, or if the jitter buffer contains a large number of packets.

Figure 17 shows a test scenario. The network consists of 4 iPAQs. The routing protocol used is AODV. During the test, the packets were extremely delayed, as shown in Figure 18. The late packets arrived after 4.5 seconds. The software client decides to flush the buffer to go back to the original line. Figure 19 shows the time line of this test scenario. The VoIP software client knows that the packets are 4.5 seconds late since it keeps track of the original time line. Nevertheless, the VoIP software client decides to play them anyway even though they are very late, because all the packets in the jitter buffer are late and there are no better alternatives.

When the VoIP software client receives packets after a normal delay, it decides to drop all the old packets to catch up with the sender.

Figure 17 Network example

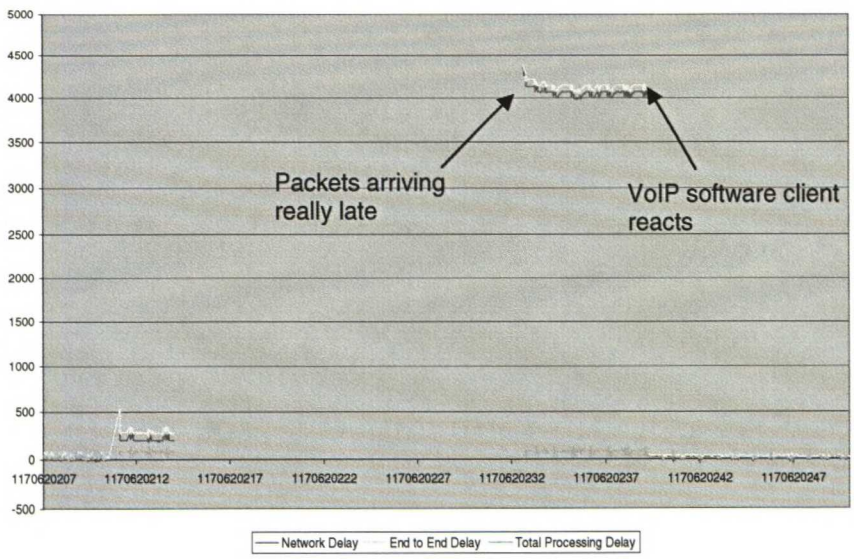


Figure 18 delay example

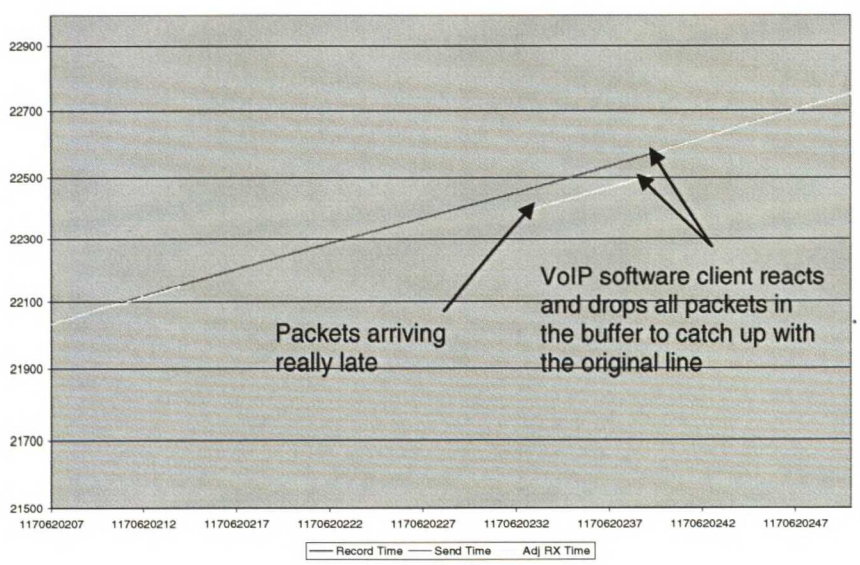


Figure 19 Time line example

2.3.3. Silence Detection

Another mechanism allowing the receiver part to keep the end to end delay low involves cooperation from the sender part. This is done by silence detection. Silence detection is made possible only in voice conversations. Silence detection exploits the fact that in a human dialog, one person speaks and the other one listens.

There are three major components included in silence detection [18],

- Voice Activity Detection (VAD): the sender part measures the energy of recorded samples, and decides whether to send or not. The sender part must be fast enough not to miss the first part of a spoken sentence or include the trailing silence after the sentence. This method adds additional delay because it requires a look-ahead buffering to calculate the energy amount.
- Discontinuous Transmission (DTX). If VAD detects silence, DTX will stop packet transmission. This is also useful for maximizing battery life if the user has a battery power node.
- Comfort Noise Generator (CNG). During a silence period, if the receiver part's speaker is turned off, the user may believe that the call is disconnected and may attempt to end the call. CNG helps regenerating background noise in the receiver part to prevent the user from thinking that the call has ended.

In this project, the VAD algorithm was simplified to measure the peak value of the recorded frame, not the energy level. The algorithm is further optimized not to examine the whole recorded buffer. This optimization is necessary since the iPAQs have limited processing power. Practical trials to implement energy calculations significantly reduced the VoIP software client performance.

The main optimizations used in the silence detection algorithm implemented in the VoIP software client are:

- Examine every 4 bytes, since at least 4 bytes of activity will be recorded when a letter is spoken.
- The algorithm returns once it finds the first value above the threshold value.

In the context of audio management, silence detection helps the receiver to keep end to end delay low, hence, preventing the receiver from flushing its buffer.

If the recorded audio level is below a certain threshold, the sender will assume that it is a silence period and will not send anything. The receiver part will drain its jitter buffer normally until a buffer under run occurs. When the silence period is over, the sender part resumes sending packets again. At this point, the receiver sees this event as if it is a new audio stream. Figure 20 presents what happens to the realtime analysis when silence detection is used.

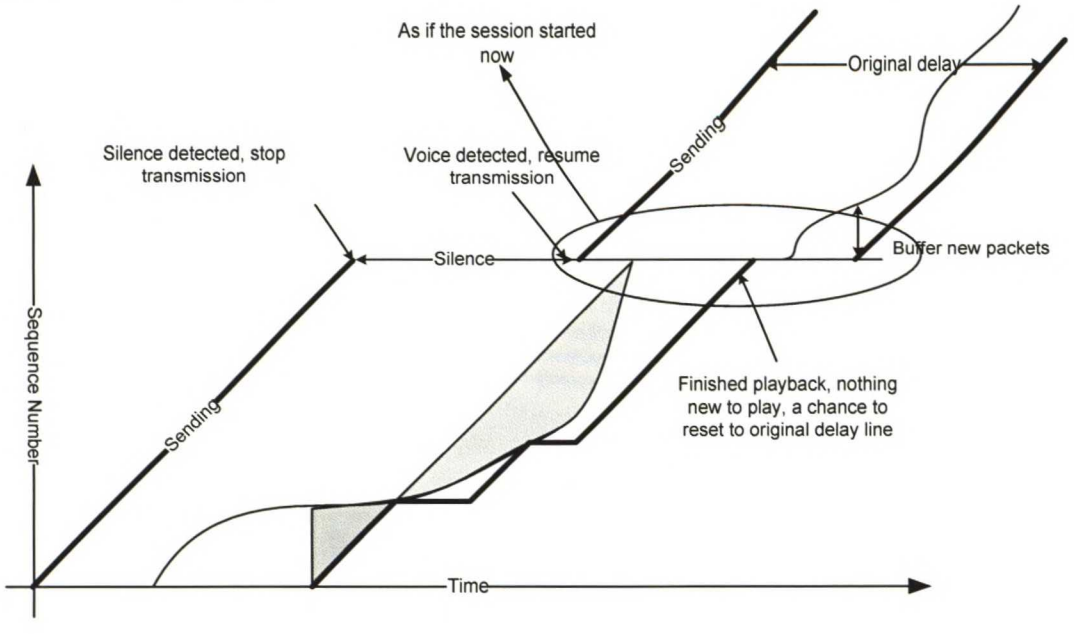
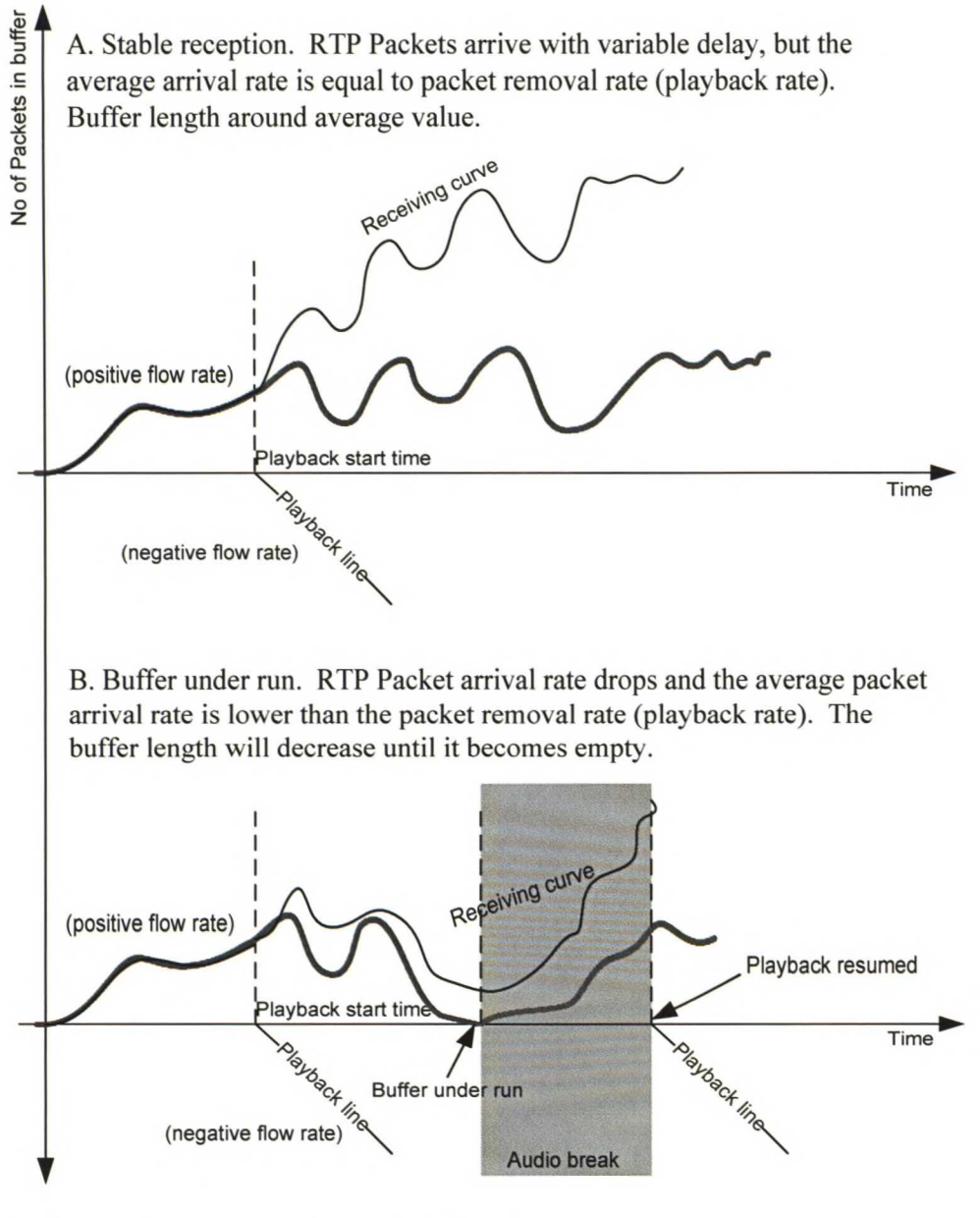


Figure 20 Timing diagram and silence detection

When the transmission is stopped during a silence period, information about the silence period is not sent from the sender part to the receiver part. Instead, the receiver part is allowed to run out of buffered packets.

2.3.4. Jitter buffering

Jitter buffer is a counter measure against variable packet delay. Figure 21 presents the jitter buffer implemented in this project, where packet arrival and removal (playback) rates are drawn.



- Positive flow: packets inserted to buffer
- Negative flow: packets removed from buffer (for playback)
- The bold red line represent the number of packets in the jitter buffer. It is also the summation of both input and output flows

Figure 21 Jitter Buffer Length

If the jitter buffer is large, it will take longer to become empty, and audio will not be broken since the VoIP software client is able to sustain the output playback rate from the jitter buffer.

However, this also means holding RTP packets for a longer period of time in the jitter buffer, which will increase the end to end delay.

2.4. Voice Quality in VoIP

The term Quality of Service (QoS) refers to the mechanisms by which a certain level of performance and quality of data stream is guaranteed. Guaranteed quality level can be achieved by methods such as:

- Allocating bandwidth in a channel.
- Allocating space in a router memory.
- Assigning priority in a routing algorithm.

Applications are classified according to their QoS requirements. 3GPP classifies applications into four classes [19]:

1. Conversational. This class has the highest real time requirement. It is sensitive to delay, but insensitive to data loss. Conversational class applications require minimum end to end delay. Examples of conversational applications are VoIP and videoconferencing.
2. Streaming. This class requires low jitter, synchronization, and good bandwidth. Applications of this class can tolerate longer delay than conversation class applications. Streaming applications buffer larger amount of media data to alleviate jitter or narrow bandwidth conditions. Examples of streaming class applications are TV and Radio station streaming.
3. Interactive. This class is more sensitive to data loss than delay. Interactive class applications, such as web browsing, make a request and expect the response within an acceptable delay which does not sacrifice interactivity. If the response comes too late, the user may abort the request.
4. Background. Applications of this class are least sensitive to delay. In fact, data delivery can happen after few hours. Examples of background applications are short messaging service, and file download. Applications of this class require correct data delivery.

Figure 22 illustrates each class dependency on delay and data loss.

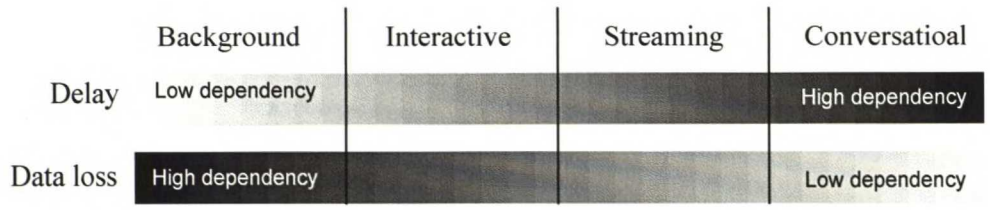


Figure 22 Quality of Service classes

The ability to predict the class of applications that will run on a network will allow for optimizing the network for the applications use. If the network is primarily used for conversational class applications, the design and layout of the network is driven by the requirements of low end to end delay. Therefore, the network is designed with less number of hops in the backbone. The routers used in this kind of networks have smaller buffers and higher processing capacities.

On the other hand, if the network is used for background traffic applications, the network is designed to achieve guaranteed data delivery and correctness. Therefore, redundant network elements are added and additional protocols are used to verify data correctness, such as using TCP instead of UDP.

The most common mechanisms to guarantee QoS level in a network are the use of Differential Service (DiffServ) [23] and Integrated Services (IntServ) [24] with Resource Reservation Protocol (RSVP) [25].

IntServ reserves resources, such as processing capacity, priority or buffer space in the routers of the network between the two ends. IntServ guarantees the availability of needed resources to fulfill the QoS requirements of the data class by booking resources in advance. IntServ uses RSVP to signal resource reservation across the network.

DiffServ utilizes the Type of Service (ToS) field in the IP packets to indicate the class of data. ToS is also called Differentiated Services Code-Point (DSCP). A DiffServ compatible router will recognize this field and act accordingly.

In Ad hoc networks, it is not possible to predict the class of applications that will be used in the network. It is also not possible to lay out a structured topology and therefore it is impossible to reserve resources.

In wireless LAN, the current versions of WLAN MAC protocols do not support QoS mechanisms. However, IEEE 802.11e is designed to support QoS in WLAN environments [26]. In this project, the WLAN environment is a best effort network using the IEEE 802.11b MAC

protocol. All nodes are equipped with 802.11b WLAN cards operating at 2.457 GHz frequency at 11 Mbps bit rate.

2.4.1. Quality of Service Factors

Packet loss

Packet loss in wireless networks is very common. Common examples of events that can lead to packet drop are router buffer overflow, router crash, broken link or high interference on the radio interface in case of WLAN networks. The event that leads to packet loss is random, but it usually results in a number of packets in succession to be lost.

Conversational class applications are less sensitive to packet loss than background class applications. Conversational class applications do not try to guarantee successful packet delivery, and therefore these applications use the UDP/IP protocol.

The VoIP software client designed for this project can ignore two lost packets at most. It will compensate for the lost packets by repeating the previous packet, ensuring continuous playback and no audio breaks. It must be mentioned that even though the audio breaks are short, they are perceived by the user as a popping sound.

A larger number of lost packets will not be compensated for, and audio in this case will be broken. The audio break will be detectable, and eventually the audio session will be of low quality. The only remedy for the lost audio is for the application user to repeat what was said. This can be considered as a retransmit operation done by the application user.

Jitter

Packets arrive at the receiver after some delay. The delay between sending and receiving packets consists of a fixed part and a variable part. The fixed delay part is due to packet transmission on the physical channel. The variable delay part is due to the current network topology and software layers load condition.

By denoting the delay as t_D, the fixed part as L, and the variable part as J, then:

$t_D = t_{arrival} - t_{departure} = L + J.$

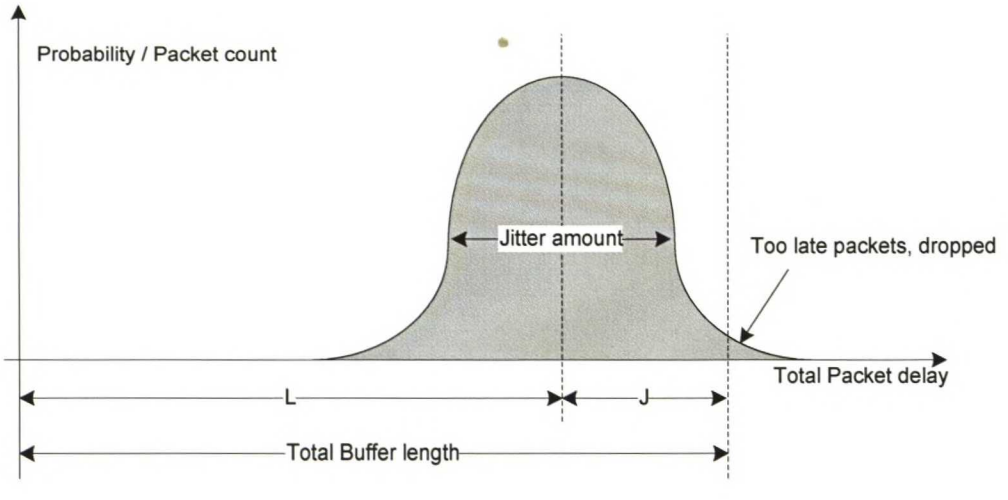


Figure 23 Jitter distribution

Figure 23 shows a distribution of how packets are expected to arrive. The jitter amount is the standard deviation of the distribution shown in Figure 23. Less jitter means narrower distribution and more predictable packet arrival.

End to end delay

The ITU-T standard G.114 [3] recommends that one-way end to end delay should be kept below 150 ms for an acceptable quality. The end to end delay ranging from 150 to 400 msec is still acceptable, but a larger delay than 400 msec is unacceptable. Increasing the length of the jitter buffer will reduce the effect of variable delay, but this will also increase the end to end delay. On the other hand, reducing the jitter buffer will result in a low quality session with a lot of pauses in the playback.

Figure 24 shows the lifetime of an audio instance, from the moment it is recorded at the sender part to the moment it is played back at the receiver part.

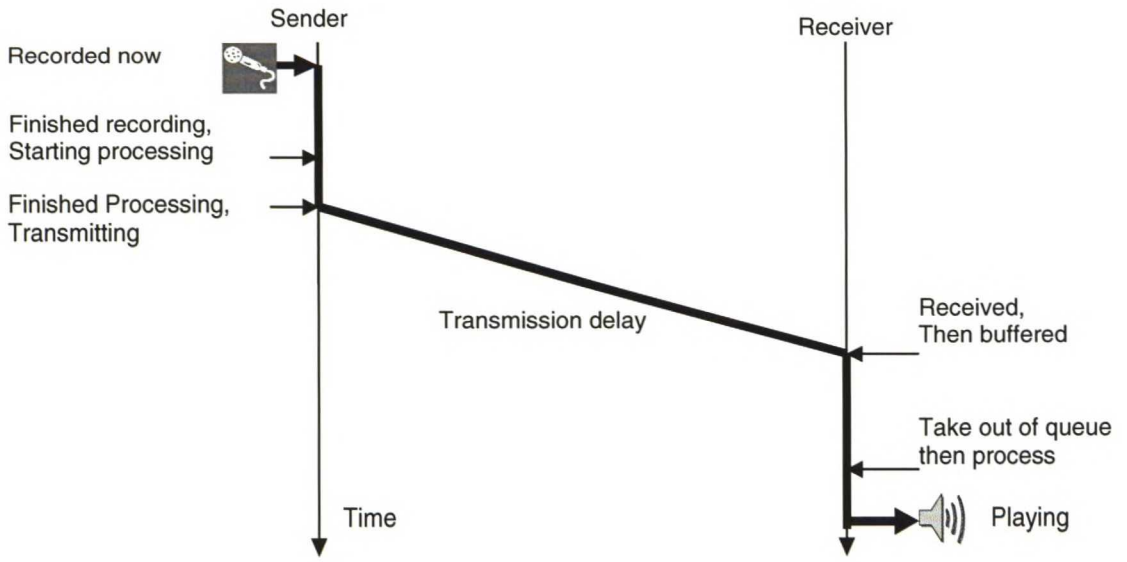


Figure 24 End to End delay

Out of order packet delivery

When a stream of packets is transmitted in IP networks, different packets may follow different routes. The result is that packets may arrive out of order at the receiver part.

The VoIP software client correctly re-orders arriving packets in the jitter buffer. Additionally, The VoIP software client assigns an arrival deadline for each packet once playback has started. This assignment is based on the RTP packet sequence number. If an RTP packet arrives out of order and later than the assigned deadline, the VoIP software client will drop it.

Routing Protocol

As discussed in section 2.1.4, the two routing protocols fulfill some but not all VoIP real time requirements. In this project, the proactive routing protocols; OLSR, and the reactive routing protocols, AODV will be compared under different scenarios.

Chapter 3

3. VoIP Software Client

In this project, a VoIP software client application is specially designed and implemented. This application is intended to be used as part of the test bed, and therefore it is possible to change it to suite the needs of testing.

The VoIP software client must fulfill the following requirements:

- Setup a call using SIP protocol signaling.
- Be able to record and playback audio.
- Encode and decode audio data using GSM codec.
- Transport encoded audio data using RTP protocol.
- Measure and log RTCP and performance data for analysis.
- Ability to fine tune the client parameters in order to study the effect of the parameters on the quality of service. Such parameters include the number of GSM audio frames per RTP packet, the jitter buffer length and the audio device specific parameters.

3.1. VoIP Software Client Design

This section details the VoIP software client design. The VoIP software client is implemented in C and C++. Linux shell scripts are also included to facilitate user operations. Figure 25 shows the VoIP software client high level block diagram. There are 4 major components: the SIP component, the RTP component, the logging component and the audio component.

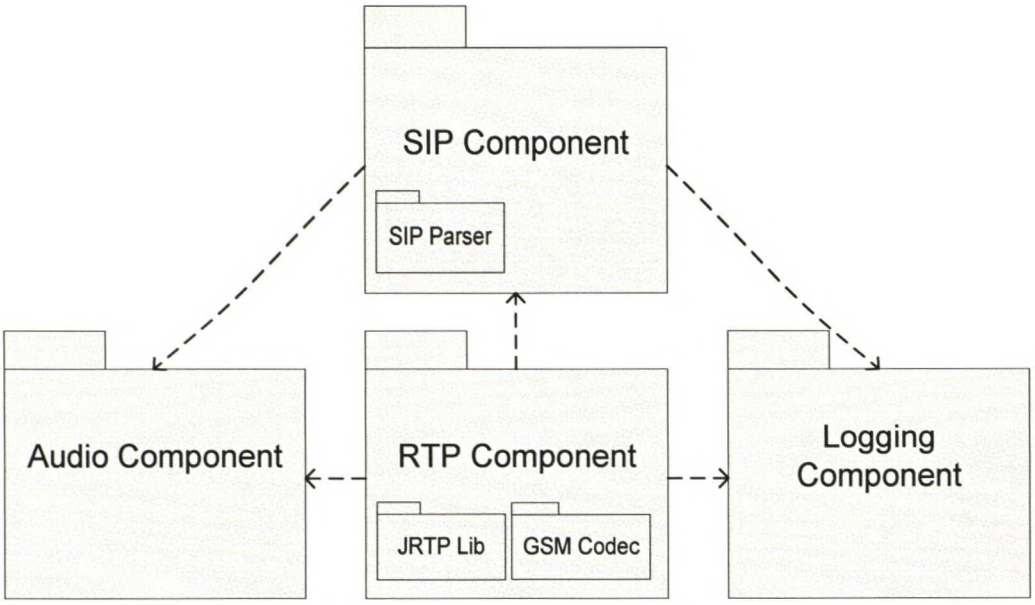


Figure 25 VoIP software client software block diagram

The SIP component is written in C because the open source SIP library used is written in C. The RTP component is written in C++ because the open source RTP library (JRTP) [20] is an object-oriented library written in C++. Interaction between SIP and RTP components happens through C wrapper functions.

The SIP component contains most of the logic. It decides how to execute user commands, how to respond to received SIP messages, and when to start or stop RTP media transportation.

The RTP component is responsible for recording and playing back audio data, encoding and decoding GSM audio frames, and transmitting and receiving RTP. The RTP component collects performance metrics from RTPC reports. The RTP component also logs the time of all the operations done. The timing information collected is used to analyze the delay budget.

The audio device component is a wrapper around the audio device hardware. It shields its complexity and exposes a simple interface. The audio device component also assists the SIP component to manage multiple accesses to the audio hardware device.

Finally, the logging component is responsible for writing log files in a specified format. Other components request writing collected information to log files through the logging component interface.

3.2. SIP Component

Figure 26 illustrates a detailed diagram of the SIP component. The SIP component consists of 3 threads: the receiving thread, the processing thread and the UI Thread. The receiving thread listens to the SIP socket. It parses and verifies received SIP packets. If the received SIP packets are valid, the receiving thread will push them on the SIP packets queue.

The UI thread is responsible for reading the VoIP software client user input. It parses and packs the input into command structures and then it pushes them on the command queue. The UI thread also polls the message queue to print what it finds there to the user.

The processing thread takes input from SIP packets queue and command queue. The processing thread implements session management and is responsible for taking appropriate actions.

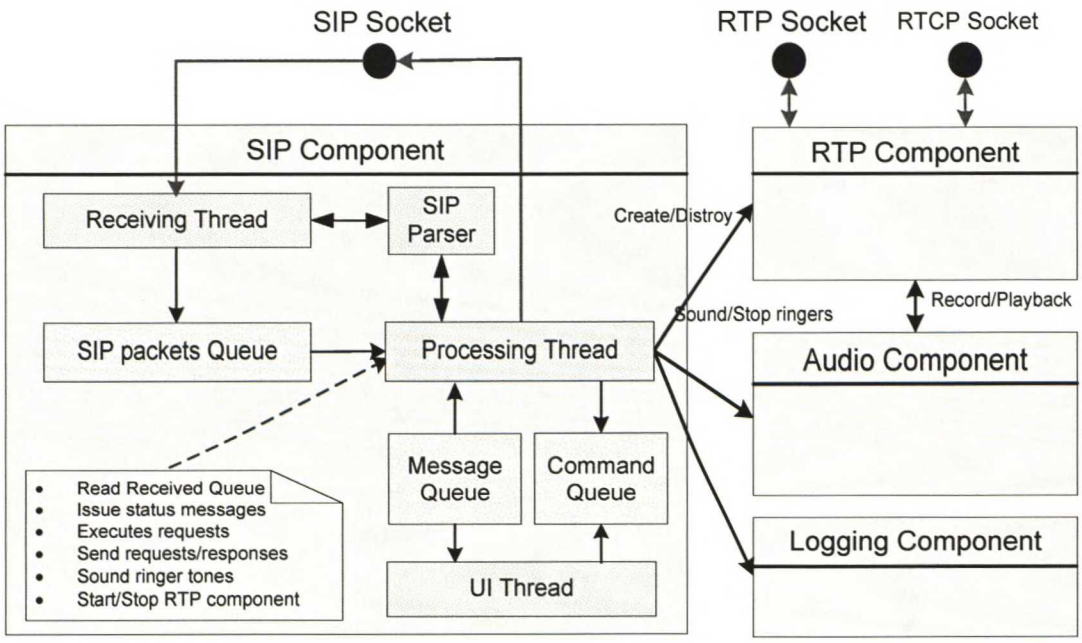


Figure 26 SIP component block diagram

3.2.1. Modules in SIP Component

SIP Component Sockets

This is the socket the processing thread opens for SIP signaling. RFC 3261 [4] recommends it to be port 5060. However, the port number can be changed in the configuration file.

SIP Component Receive Thread

The receive thread will keep listening and for whatever is received, the receiving thread will:

- Validate the received packet to be a valid SIP packet using the SIP parser library.
- Fill a SIP packet structure.
- Place that SIP packet structure on the SIP packets queue.

SIP Component Packets Queue

The SIP packets queue is a first-in first-out (FIFO) queue holding SIP packets. The receiving thread will push received SIP packets on it, and the processing thread will pop out from it. This queue facilitates the asynchronous communications between the receiving thread and the processing thread.

SIP Component Command Queue

The command queue is a FIFO queue that holds command structures. The UI thread will parse and check commands entered by the VoIP software client user, and then it fills a command structure and places it in this queue. The possible commands are INVITE, MESSAGE, SESSION, and SESSIONS. Refer to appendix 8.2 for more information.

SIP Component Message Queue

The message queue is a FIFO queue. Anything the processing thread wishes to delegate to the user is placed in the message queue. Messages can be prompts about incoming call or a received SIP MESSAGE packet to be printed to the user.

3.2.2. VoIP software client sequence diagram

Figure 27 presents the sequence diagram of the SIP component. It also gives an idea about how the main threads are started.

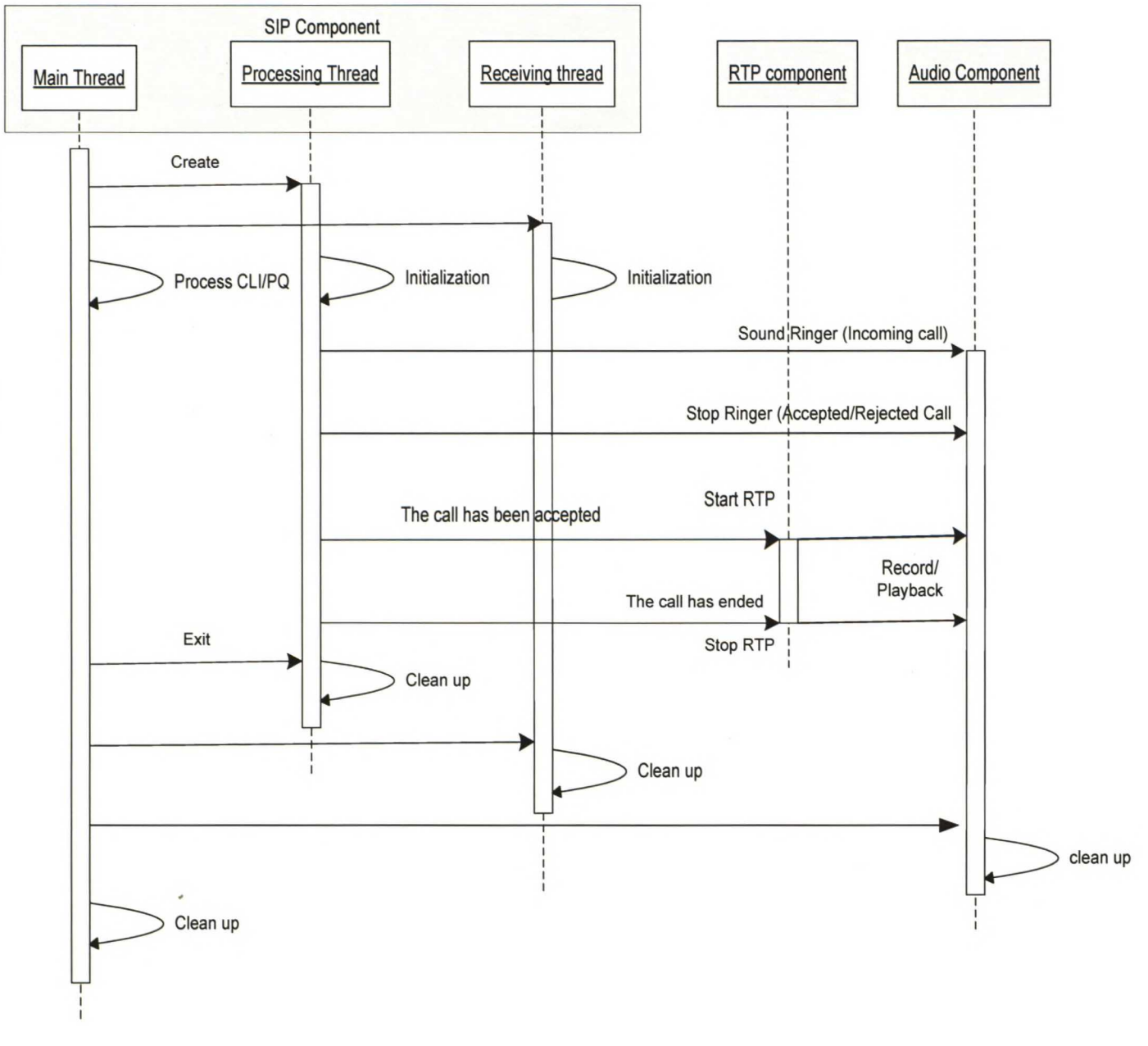


Figure 27 VoIP software client sequence diagram

3.2.3. SIP Component Session Management

A SIP dialog can be uniquely identified by a combination of the "TO" tag, the "FROM" tag, and the "CALL ID" [4]. When the caller user agent initiates a call, it generates a "FROM" tag and a

"CALL ID" and includes them in the initial INVITE message. If the callee accepts the call, the callee user agent will send 200 OK response including a "TO" tag. In this project, since conference calls are not supported, uniqueness is guaranteed only between the two communicating parties.

Session management is implemented as an in-memory database holding the following information:

- StateCode: the current state code of the session.
- SIPMethod: the SIP method request triggering the state change. This keeps track of what request caused the session to end in this state.
- LocalSeqNum: the last sequence number of the last issued request, as specified in RFC 3261 [4.]
- RemoteSeqNum: the last remote sequence number of the last request received, as specified in RFC 3261 [4.]
- CallID: the "CALL ID" of the current call.
- FromTag: the "FROM" tag of the current call.
- ToTag: the "TO" tag of the current call.
- RemoteURL: the other party URI. Since there is no SIP infrastructure in this project, this field holds the IP address of the remote party.
- SDPBody: information exchanged in SDP during session setup.
- TimeStamp: the time when the session was created.
- Secure: a flag indicating if the sessions is secured or not.
- Route Set: a list of intermediate nodes that the SIP messages should follow.

Although Secure and Route Set are specified by RFC 3261 [4], they are not implemented in this project. The reason is that Route Set is not needed in ad hoc networks, as there are no SIP proxies in the middle to specify as route alternatives. Route set does not refer to possible routes between the two end nodes in the ad hoc network. In ad hoc networks, the VoIP software client is unaware of existing routes through different intermediate nodes, since this is handled by the lower transport layer (IP), aided by the routing protocols (AODV and OLSR). Security is not included since it is not the focus of this project.

When a user wants to make a call, the processing thread builds a session containing all required information, except the "TO" tag field, which is supplied by the callee when accepting the call. The processing thread prepares and sends a SIP INVITE request message to the callee IP

address. If the user wishes to cancel the call, the processing thread will immediately send a CANCEL request and then destroy the related session.

If the call is not canceled, the processing thread, upon receiving a response, will look up the related session using the "FROM" tag, the "CALL ID", and if present, the "TO" tag. It will then consult the session state management logic for an appropriate action. Figure 28 shows the session management state diagram.

If no matching session is found, and the received SIP packet is not an INVITE message, a 488 NOT ACCEPTABLE HERE message will be replied.

Session Management State machine

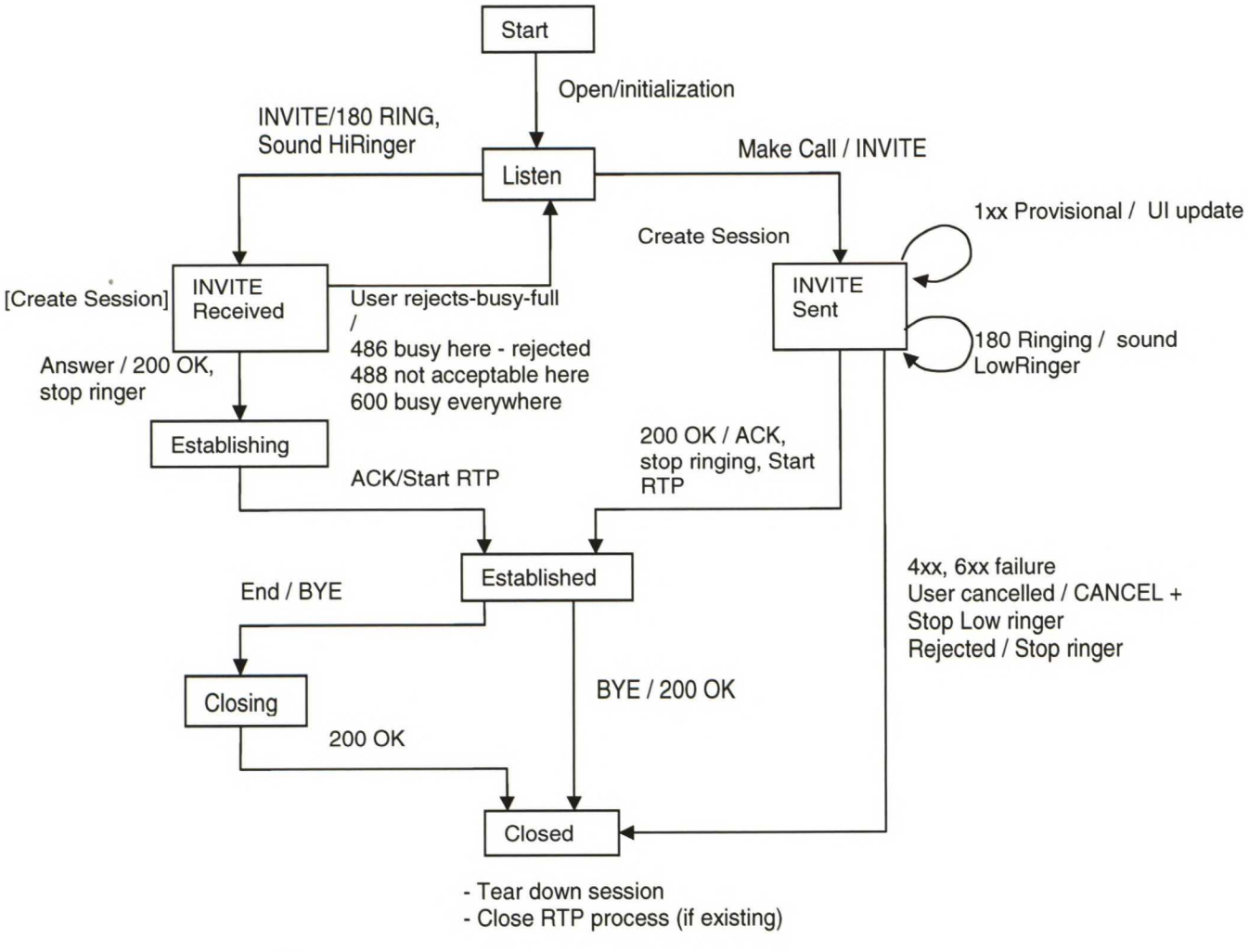


Figure 28 SIP component session diagram state diagram

In Figure 28, the blocks represent the current state of the session. The arrows show the input event and the output action, and they point to the destination state.

Table 4 shows the state changes when a caller wants to make a new call, which the callee accepts.

Table 4 State transition example: establishing call

Current State	Input event	Output action	Next State
Listen	Make Call	Send INVITE message	INVITE sent
INVITE sent	1xx provisional responses	Update UI if necessary, stay in same state	INVTE sent
INVITE sent	180 RINGING provisional response	Instruct audio component to sound the status ring back tone	INVITE sent
INVITE sent	200 OK	Send ACK, stop the status ring back tone. Start RTP module.	Established

Table 5 shows the state changes when a callee receives a call.

Table 5 State transition example: ending call

Current State	Input event	Output action	Next State
Listen	SIP INVITE request received	Sound the notification ringing tone. Send 180 RINGING, and create a session.	INVITE received
INVITE Received	User picks up (using UI)	Send 200 OK, stop the notification ringing tone	Establishing
Establishing	ACK received	Start RTP	Established

On going conversation. Then the user wants to end the call.			
Established	User select to end call (using UI)	Send BYE	Closing
Closing	200 OK received	Do nothing	Closed
Destroy the session, release audio device and clean up resources.			

It is worth mentioning that except for SIP MESSAGE messages, all other messages go through session look up stage. If no related session is found, 488 NOT ACCEPTABLE HERE is replied.

Only INVITE messages cause creation of new sessions. Termination of a session is with a legitimate CANCEL or BYE messages. Note that receiving 488 NOT ACCEPTABLE HERE will also terminate a pending related session, and no reply is sent back.

3.3. RTP Component

The RTP component handles recording and playing back audio, encoding and decoding using the GSM codec [17] and data transmission using the RTP library. This component runs two threads, a sending thread and a receiving thread. Figure 29 illustrates the RTP component design.

RTP and RTCP sockets are opened and handled in the RTP library, but the RTP socket number is specified by the RTP component. RTP socket is arbitrarily selected from the available sockets on the system, such that it is an even numbered socket. The succeeding odd numbered socket is used for the RTCP socket.

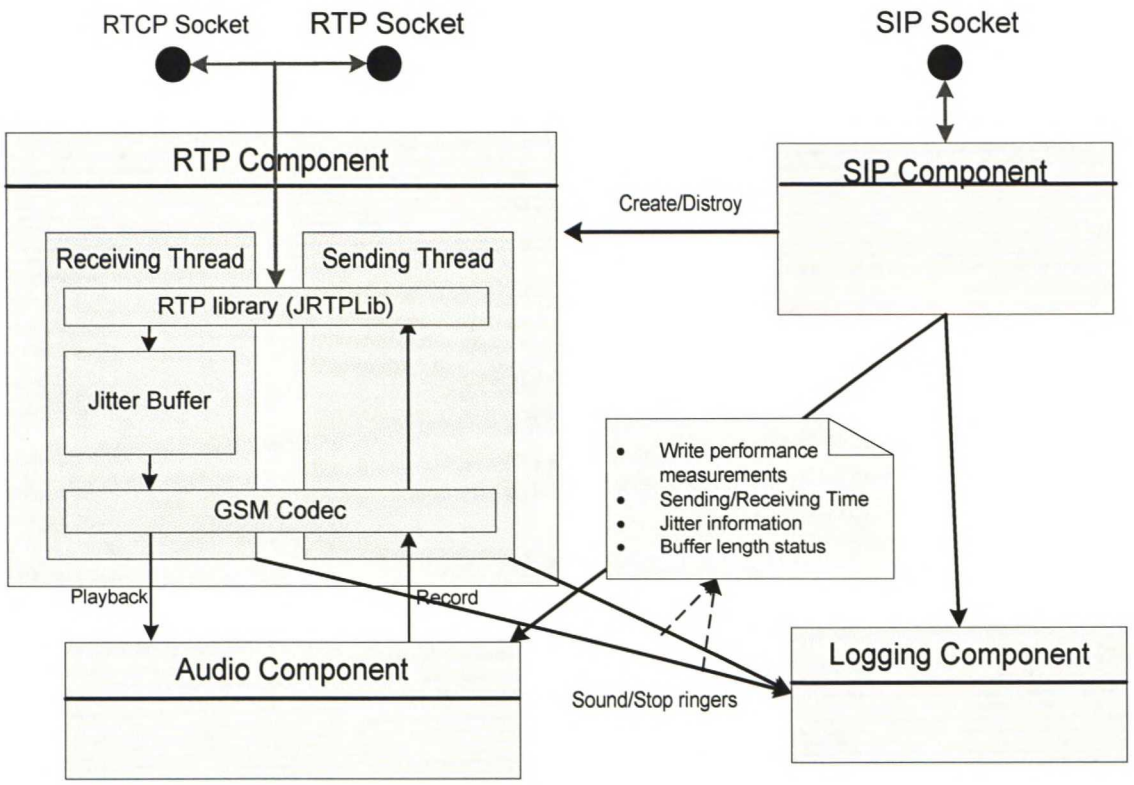


Figure 29 RTP Component design

3.3.1. Modules in RTP Component

RTP Component Sending Thread

The sending thread records audio, encodes it and then sends it in RTP packets. It also handles timing issues such that a constant rate flow of packets is maintained. The timing management designed in the RTP component is directly dependent on the nature of the used codec; which is the GSM codec, refer to section 2.3 for real time audio management.

The VoIP software client is configured to send 3 GSM audio frames in one RTP packet. Therefore, the sender thread records 60 msec of audio to create 3 GSM frames. The sending thread then packs the 3 GSM frames in 1 RTP packet and sends it on the RTP socket.

To maintain a continuous fixed packet sending rate, the sending thread records audio in parallel to encoding and sending operations. The sending thread is restricted by the timing requirements of GSM codec i.e. it has to complete a cycle every 60 msec. This is referred to as the sending time budget. Figure 30 illustrates the parallel processing of the sending thread.

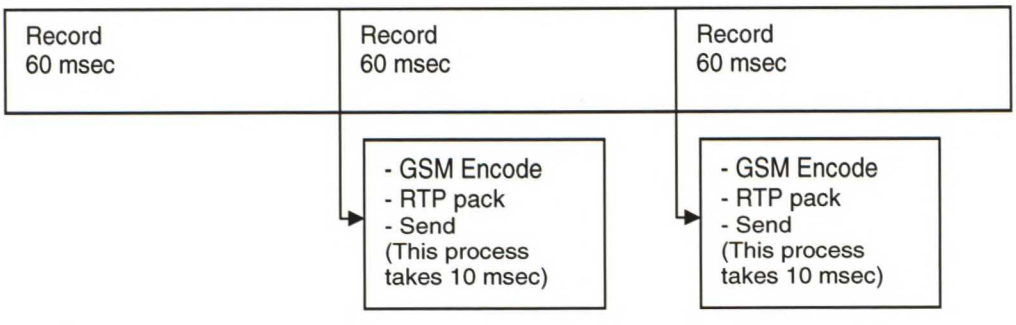


Figure 30 Parallel operation of VoIP software client

Better performance is achieved if more GSM frames are loaded in one RTP packet. Having more GSM frames in the RTP packet will reduce the share of RTP headers in the transmitted data. On the other hand, increasing the number of GSM frames sent in every RTP packet will increase the delay since the frames need to be recorded in advance. After field testing, it was found that a configuration of 3 GSM audio frames in one RTP packet gives the best performance. Figure 31 shows the RTP packet.

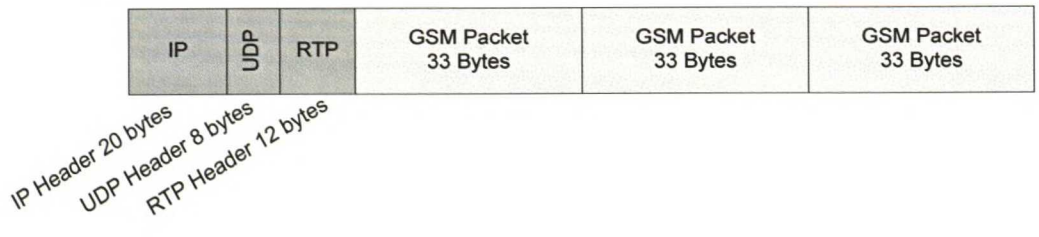


Figure 31 VoIP software client RTP packet

RTP Component Receiving Thread

The receiving thread will handle reception, buffering, decoding and playing back of audio data. The receiving thread continuously decodes and plays back buffered GSM frames at a fixed rate, which is 1 GSM frame every 20 msec. If the jitter buffer becomes empty, playback is stopped until enough packets are buffered, which will counteract jitter and help in sustaining a fixed playback rate.

The jitter buffer length is calculated by taking into account the initial jitter buffer length as specified in the configuration file, in addition to the amount of jitter reported in the RTCP

sender and receiver reports. Jitter buffering management is implemented using a state machine. Figure 32 shows the jitter buffer state machine diagram.

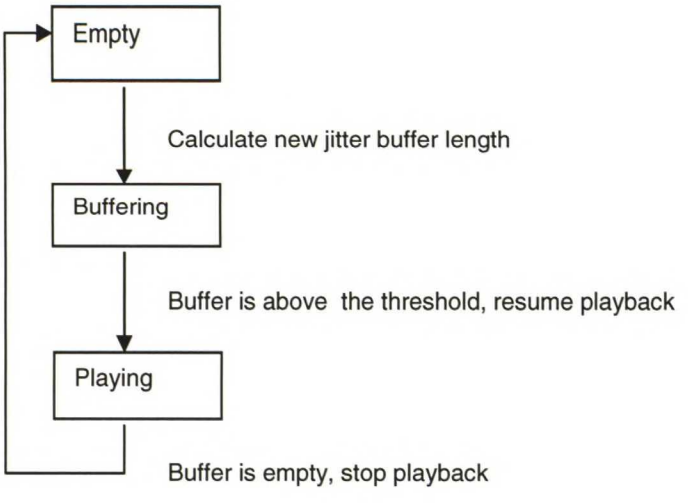


Figure 32 Jitter buffer state diagram

3.4. Audio Component

The audio device component wraps the complexity of communicating with the audio device hardware. It manages initializing, opening and closing the audio device driver and handles synchronization issues with the audio device driver. The audio component exposes a simple interface for the SIP and RTP components.

When the VoIP software client starts, the audio device is not opened. When the first incoming call or outgoing call request is made, the SIP component needs to playback ringing tones. At that point, the SIP component will request from the audio component to open and initialize the audio device. Once the audio device is open, it stays open until the SIP component request closing it from the audio component.

When the audio device component is initialized, some information about the audio device status is printed:

```
######## AUDIO DEVICE REPORT ########
        Setting audio format
        Setting mono
        Setting sampling speed to 8000
        Setting full duplexing
        Audio fragment size: 1024 bytes
```

```
        Input buffer total fragments: 0
        Input buffer fragment size: 1024
        Output buffer total fragments: 4
        Output buffer fragment size: 1024
####################################
```

The audio component interfaces with the audio device driver through the audio system model defined by Open Sound System (OSS) [1]. OSS provides audio buffers for playback and recording. They are filled sequentially upon request. The VoIP software client code specifies how many buffer segments OSS should use and how large each buffer is.

The situation is simple when setting up buffer segments for playing back. The audio device driver will correctly handle playing back using any buffer segment setup. The audio device driver does not force writing complete buffer segments before playing back a segment content. It is capable of playing back a partially filled segment. Therefore, the VoIP software client will write the audio data to any playback segment as soon as the audio data is available.

During recording phase, the audio device driver needs to completely finish recording a buffer segment before it can return the recorded content to the VoIP software client. Therefore, it is important to keep the audio buffer segments small so as not to increase the delay overhead of recording audio. However, OSS recommends not using very small buffer segments, since this will increase the overhead of API calls, which will eventually lead to popping sound in the recorded audio.

The best audio device buffer segment setup for recording as observed from field testing is 4x512 bytes audio buffer segments. To reduce the complexity of the VoIP software client, the playback buffer is also setup in the same manner. It must be mentioned that playback buffer segments are different from recording buffer segments. Figure 33 illustrates the audio buffer segments used in the VoIP software client.

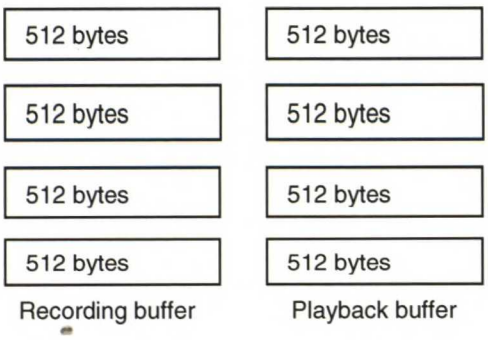


Figure 33 Audio device buffer segments in playback and recording cases

The GSM codec library handles the incompatibility between its 13 bits per sample requirement and the iPAQ audio device ability to record 16 bit samples. The GSM library ignores the least significant 3 bits of each recorded sample.

Since each GSM audio frame represents 20 msec of audio, this gives:

20/1000 (sec) * 8000 (sample/sec) = 160 (sample)

Each sample is 16 bits wide, hence the block of bytes to be requested from the audio device each time a GSM audio frame is to be sent is 16 bits * 160 = 320 bytes.

The VoIP software client performs best if 3 GSM frames are packed in one RTP packet; therefore 60 msec is recorded, yielding 960 Bytes.

3.4.1. Audio Component Ringing Tones

The audio component also implements ringing tones. There are two kinds of ringing tones:

- Notification ringing tone: This is the tone the callee hears when a call is received. This is a loud tone for notifying the user of an incoming call.
- Status ring back tone: this is the tone the caller hears while waiting for the callee to pick up. It is a low volume tone that gives feedback on the current status of signaling. It gives an indication if the callee is busy or if the call is rejected.
- Dial tone. It is the tone the caller hears when first dialing the callee number. It gives feedback to the caller about the availability of the service. It is used in the PSTN system, but not in VoIP or GSM systems.

The status ring back tone frequency and timing are country specific. Each country has standards specifying the frequency of the tone and the timing of the silence periods. This project adheres to IETF draft [21]. Only status ring back frequency and timing of Finland are implemented. Unfortunately, the draft does not specify details on busy tones. The closest source for implementing the status ring back tones is found in GenDet [27]. Table 6 shows the specifications the VoIP software client implements.

Table 6 Status ring back tones specifications

Status ring back tone	Frequency	Sounding time	Silence Time
Ringing (callee is being notified)	425 Hz	1 second	4 seconds
Busy tone	425 Hz	0.5 second	0.5second

It is worth mentioning that dial tone consists of 350 Hz and 440 Hz harmonics [22] , but it is not used in systems such as GSM or VoIP.

3.5. Logging Component

This component is responsible for formatting and writing collected data measurements during an audio call. The VoIP software client will collect the following measurements:

- The time when the audio data is recorded.
- The time when the RTP packet is sent.
- The time when the RTP packet is received.
- The time when Audio data is played back.

The standard logs generated during an audio session are:

- Timing_tx.log: this log file contains timing information about when the audio is recorded and when the packets are sent including the sequence number of each packet.
- Timing_rx.log: this log file contains timing information about when the packets are received and the length of the jitter buffer as reported by RTCP.
- Timing_pb.log: this log file contains timing information about when the packets are played back and the current in-memory jitter buffer length.

The collected information will allow calculating important quality indicators, which are:

- The time it takes to complete audio recording and encoding.
- The time delay in the network, which is the period of time between sending and receiving times. The network delay includes the routing delay in the intermediate nodes.
- Buffering delay, which is the time a packet spends in the jitter buffer after it has been received.
- The time it takes the packet to be decoded and played back.
- End to end delay, which is the time period between when the packet is recorded and when it is played back. This is the perceived delay by the users.
- Total processing delay; which is the time difference between network delay and end to end delay. This delay includes the time the processor spends executing the VoIP software client logic, such as encoding and decoding. It also includes the buffering delay.

In addition, several other optional data can be logged. Enabling optional logging is done by compilation flags during compile time. Excluding optional logging at compile time will reduce the output binary size and speed up code execution. Such optional measures are RTCP sender and receiver reports, in addition to error and debug messages.

Chapter 4

4. VoIP Test Bed

The main objective of the VoIP test bed is to observe the effects of the quality of service factors [section 2.4.1]. During every test case, the following will be measured:

- Jitter amount.
- Packet loss.
- End to end delay.
- Routing protocol overhead and its share of the bandwidth.
- Time needed to reestablish an alternative link.

The test bed also analyzes the effect of:

- Increasing number of hops between the end nodes.
- Changing the ad hoc network topology during a call.
- Changing the intermediate nodes types, whether they are iPAQs or laptops.

4.1. Test Bed Environment

One of the most important sources of errors is the difference between the two end iPAQs clocks, which has been noticed to have different clock frequencies, causing the clocks to drift. The test bed includes several shell scripts to ensure setting the clocks of both end iPAQs to the same time at the beginning of each test case.

The test bed also includes scripts to track the clock differences. Once the data is collected, another script is run to adjust the data collected to take into account the time drift between the

clocks of the two end iPAQs. Figure 34 shows an example of the time difference between the two end iPAQs.

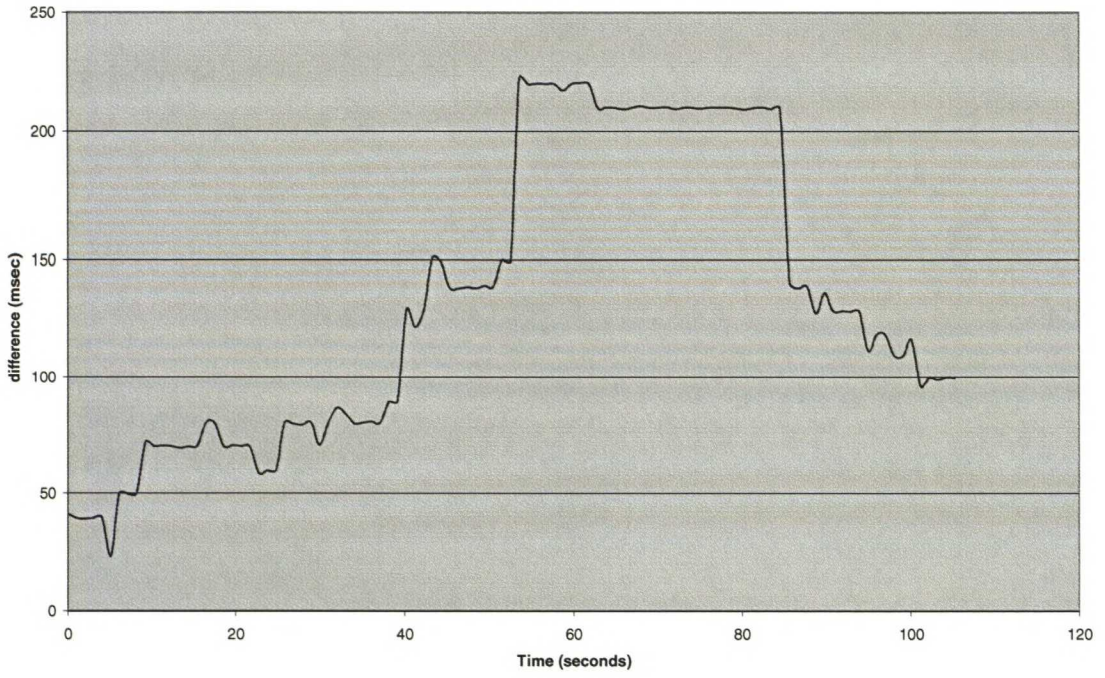


Figure 34 Time difference drift

In Figure 34, the time difference grows to over 200 msec. One of the end iPAQs will be used as a reference, while the data collected from the other iPAQ will be corrected according to this plot. Finally, the accuracy of collecting this information will directly affect the error all the test cases will have.

Another source of error is the lab environment where the test cases were carried out. There is a high WLAN activity which can start and end during a test case execution. Existing WLAN activity will congest the physical channel of WLAN. In addition, it was observed that the WLAN cards used with the iPAQs are quite unstable and need resetting regularly. A simple test setup was executed, where 2 end iPAQs and 1 intermediate routing iPAQ were used, once using AODV and another time using OLSR. Table 7 shows the AODV case, and it shows that route requests were made 54 times. Table 8 shows the same test but using OLSR protocol, it also shows that topology changes where made 55 times. This clearly indicates the unstable WLAN environment.

Table 7 AODV activity

Route Request	54
Route Reply	162
Route Error	6

Table 8 OLSR activity

HELLO	126
TC	55

Another hurdle faced in this project is the difficulty of determining the source of unexpected results, whether it is the implementation or the theory behind the implementation. For example, there are some unexpected results shown by AODV and OLSR, but it is not easy to determine if it is because of the implementation itself or because of proactive or reactive routing protocols themselves.

4.2. Test Case 1: Increasing number of hops

The objective of this test case is to measure the Qualify of Service factors as the number of hops increases. There are 6 sub test cases:

OLSR test cases:

- TC1.OLSR.2
- TC1.OLSR.3
- TC1.OLSR.4

And AODV test cases:

- TC1.AODV.2
- TC1.AODV.3
- TC1.AODV.4

The format of test case names is TC1.<Protocol Name>.<Num of iPAQs>. Finally, a summary section is included to present all the collected values in one place.

4.2.1. Test Case 1 sub test cases

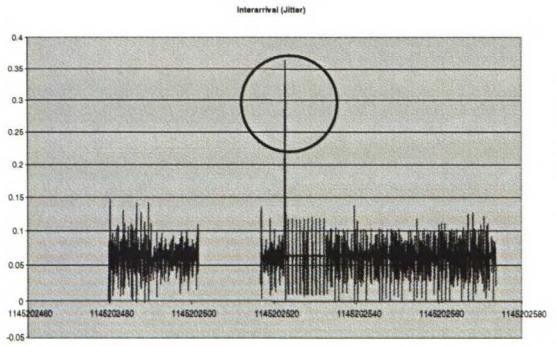
TC1.OLSR.2

In this test case, there are two iPAQs, and therefore, no routing is needed. The iPAQs are running OLSR routing protocol. The physical link will be broken in the middle of the test case to measure the routing protocol ability to reestablish the link.

<table>
<tr><th>Number</th><th colspan="2">Name</th></tr>
<tr><td>TC1.OLSR.2</td><td colspan="2">2 Nodes, 1 hop in between.</td></tr>
<tr><td colspan="2">Environment, equipment and software used:</td><td colspan="2">Client parameters:</td></tr>
<tr><td colspan="2" rowspan="4">Two 3850 iPAQs with Familiar Linux.
RTP Client version 1.0.
Tcpdump.
UU OLSR version 0.4.5.
802.11b, 11Mbps, ch 10 (2GHz.)</td><td>Initial Jitter Buffer (msec)</td><td>60</td></tr>
<tr><td>Payload length (GSM/RTP)</td><td>3</td></tr>
<tr><td>Routing Protocol</td><td>OLSR</td></tr>
<tr><td></td><td></td></tr>
<tr><td colspan="3">Topology</td></tr>
<tr><td colspan="3">  </td></tr>
<tr><td colspan="3">Execution</td></tr>
<tr><td colspan="3">Start the client software. Speak for 2 min. Break the connection for 30 seconds. Reestablish the connection for another 2 mins.</td></tr>
</table>

Plots

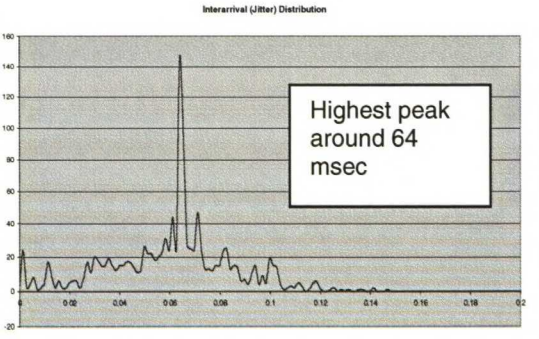
Inter-arrival (Jitter)



X-axis: time in seconds.

Y-axis: Inter-arrival amount in seconds.

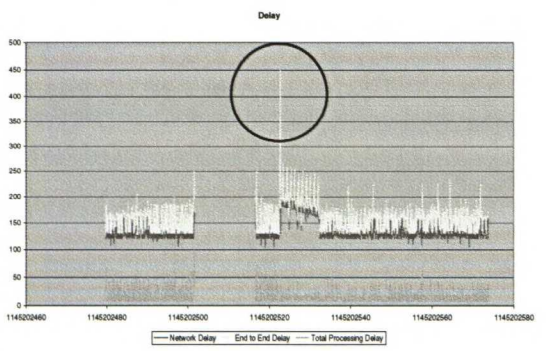
Inter-arrival (Jitter) Distribution



X-axis: Inter arrival time in seconds.

Y-Axis: Number of packets.

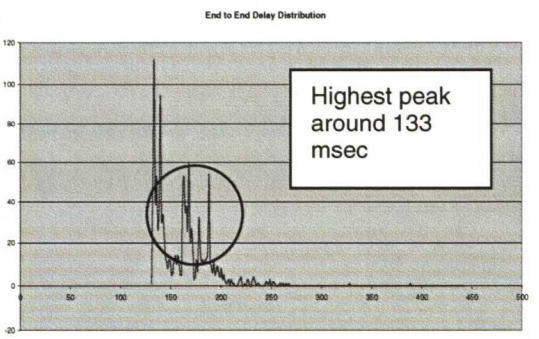
End to End Delay



X-axis: time in seconds.

Y-axis: Delay in msec.

End to End Delay distribution



X-axis: Time in msec.

Y-Axis: Number of packets.

The packets arriving later than usual appear on the end to end distribution plot as spikes above the normal end to end delay, as the circles highlight.

Analysis:

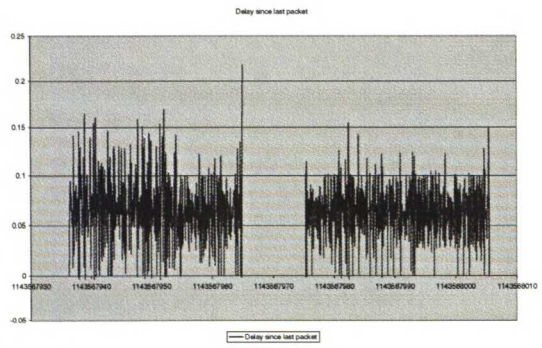
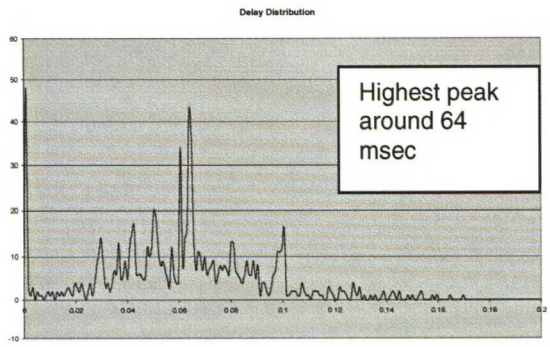
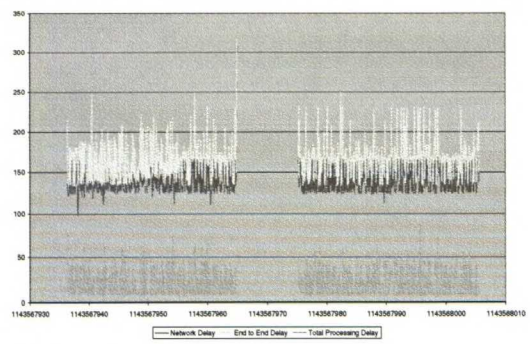
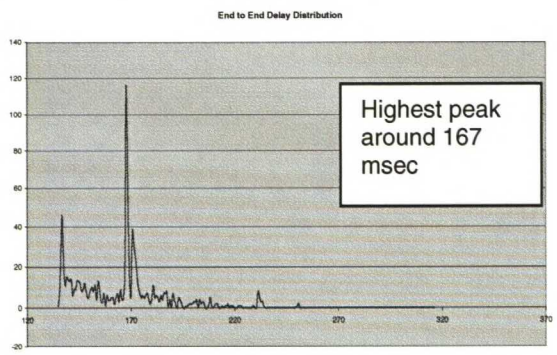
All values in msec	End to End Delay	Network Delay	Processing Delay	Inter-arrival time
Average	158	141	16	60
Std deviation	27	25	11	26
90% percentile	187	171	38	92

Packet loss (packet)	3	
Re-establish time (second)	0.5	
Protocol overhead (packets)	109 / 3215 = 3.39 %	
Protocol overhead (bytes)	7766 / 492357 = 1.57%	
Average packet size	RTP (byte/packet)	OLSR (byte/packet)
	153	72

TC1.AODV.2

This is similar to test case TC1.OLSR.2 except that AODV protocol is used. The physical link will be broken in the middle of the test case to measure the routing protocol ability to reestablish the link.

Number	Name		
TC1.AODV.2	2 Nodes, 1 hop in between.		
Environment, equipment and software used		Client parameters	
• Two 3850 iPAQs with Familiar Linux. • RTP Client version 1.0. • Tcpdump. • UU AODV version 0.91. • 802.11b, 11Mbps, ch 10 (2GHz.)		Initial Jitter Buffer (msec)	60
		Payload length (GSM/RTP)	3
		Routing Protocol	AODV
Topology			
			
Execution			
Start the client software. Speak for 2 min. Break the connection for 30 seconds. Reestablish the connection for another 2 mins.			

Plots	
Inter-arrival (Jitter)  X-axis: time in seconds. Y-axis: Inter-arrival amount in seconds.	Inter-arrival (Jitter) Distribution  X-axis: Inter arrival time in seconds. Y-Axis: Number of packets.
End to End Delay  X-axis: time in seconds. Y-axis: Delay in msec.	End to End Delay distribution  X-axis: Time in msec. Y-Axis: Number of packets.

Analysis				
All values in msec	End to End Delay	Network Delay	Processing Delay	Inter-arrival time
Average	163	143	20	61
Std deviation	21	19	14	31
90% percentile	188	165	41	99

Packet loss (packet)	1
Re-establish time (second)	0.5

Protocol overhead (packets)	170/2353 = 7.22 %	
Protocol overhead (bytes)	11544/361038=3.19%	
Average packet size	RTP (byte/packet)	AODV (byte/packet)
	153	68

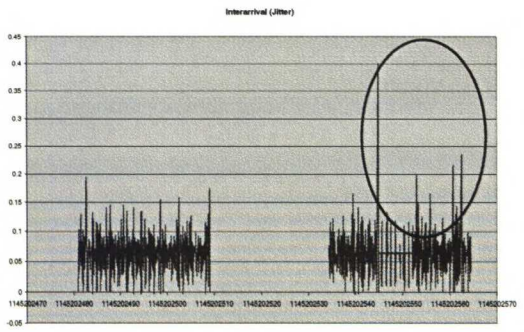
TC1.OLSR.3

In this test case, 2 hops are involved with 3 iPAQs, one of which acts as an intermediate router. All iPAQs are running OLSR routing protocol. The physical link will be broken in the middle of the test case to measure the routing protocol ability to reestablish the link.

Number	Name	
TC1.OLSR.3	3 Nodes, 2 hop in between.	
Environment, equipment and software used:	Client parameters:	
• Two 3850, one 3900 iPAQ. • Familiar Linux. • RTP Client version 1.0. • Tcpdump. • UU OLSR version 0.4.5. • 802.11b, 11Mbps, ch 10 (2GHz.)	Initial Jitter Buffer (msec)	60
	Payload length (GSM/RTP)	3
	Routing Protocol	OLSR
Topology:		
		
Execution:		
Start the client software. Speak for 2 min. Break the connection for 30 seconds. Reestablish the connection for another 2 mins.		

Plots:

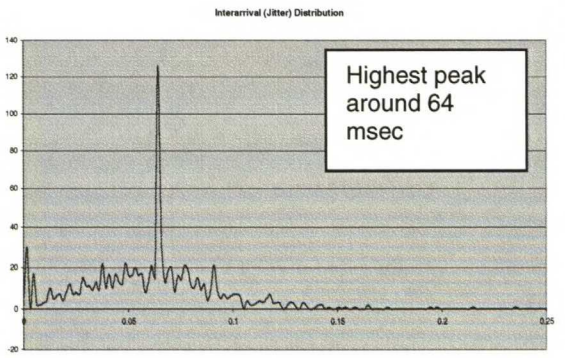
Inter-arrival (Jitter)



X-axis: time in seconds.

Y-axis: Inter-arrival amount in seconds.

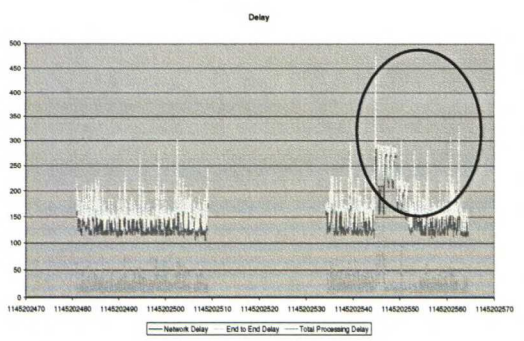
Inter-arrival (Jitter) Distribution



X-axis: Inter arrival time in seconds.

Y-Axis: Number of packets.

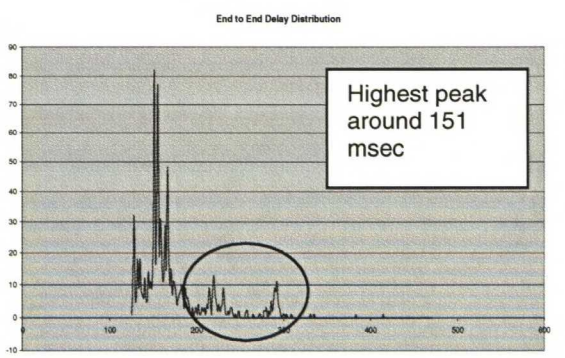
End to End Delay



X-axis: time in seconds.

Y-axis: Delay in msec.

End to End Delay distribution



X-axis: Time in msec.

Y-Axis: Number of packets.

An extra end to end delay occurs during the call. The delay distribution shows this fact. The circles highlight this event.

Analysis:

All values in msec	End to End Delay	Network Delay	Processing Delay	Inter-arrival time
Average	166	149	16	60
Std deviation	43	40	17	33
90% percentile	222	202	38	97

Packet loss (packet)	4	
Re-establish time (second)	8	
Protocol overhead (packets)	181/4688 = 3.86%	
Protocol overhead (Bytes)	13030 / 720134 = 1.81%	
Average packet size	RTP (byte/packet)	OLSR (byte/packet)
	153	72

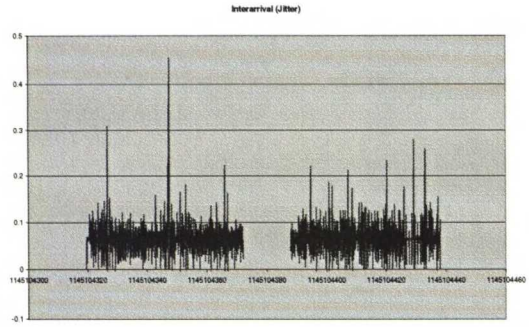
TC1.AODV.3

This test case is similar to TC1.OLSR.3. 2 hops are involved with 3 iPAQs, one of which acts as a middle router. All iPAQs are running AODV routing protocol. The physical link will be broken in the middle of the test case to measure the routing protocol ability to reestablish the link.

Number	Name		
TC1.AODV.3	3 Nodes, 2 hop in between.		
Environment, equipment and software used:		Client parameters:	
• Two 3850 and one 3900 iPAQs • Familiar Linux. • RTP Client version 1.0. • Tcpdump. • UU AODV 0.91. • 802.11b, 11Mbps, ch 10 (2GHz.)		Initial Jitter Buffer (msec)	60
		Payload length (GSM/RTP)	3
		Routing Protocol	AODV
Topology			
			
Execution			
Start the client software. Speak for 2 min. Break the connection for 30 seconds. Reestablish the connection for another 2 mins.			

Plots

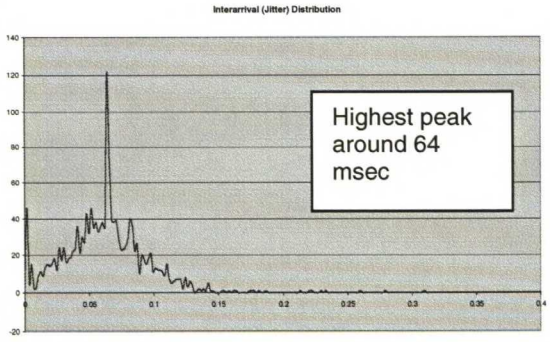
Inter-arrival (Jitter)



X-axis: time in seconds.

Y-axis: Inter-arrival amount in seconds.

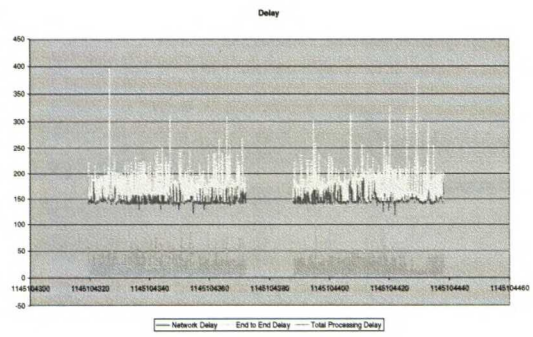
Inter-arrival (Jitter) Distribution



X-axis: Inter arrival time in seconds.

Y-Axis: Number of packets.

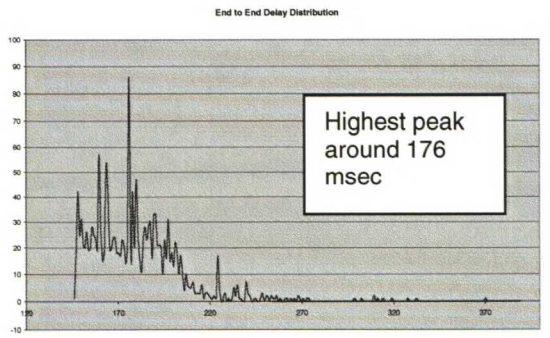
End to End Delay



X-axis: time in seconds.

Y-axis: Delay in msec.

End to End Delay distribution



X-axis: Time in msec.

Y-Axis: Number of packets.

Analysis

All values in msec	End to End Delay	Network Delay	Processing Delay	Inter-arrival time
Average	168	148	20	62
Std deviation	25	22	14	34
90% percentile	193	174	43	100

Packet loss (packet)	4
Re-establish time (second)	1

Protocol overhead (packets)	506 / 6858 = 7.38%	
Protocol overhead (bytes)	34560 / 1050198 = 3.29%	
	RTP (byte/packet)	AODV (byte/packet)
Average packet size	153	68

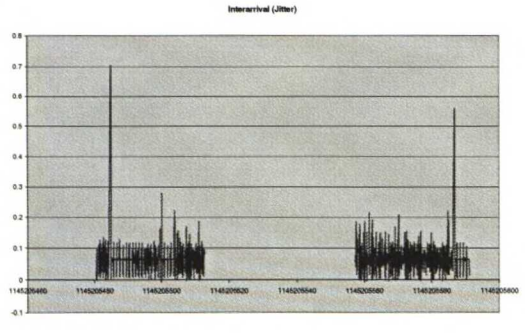
TC1.OLSR.4

In this test case, 3 hops are involved with 4 iPAQs; two are acting as intermediate routers. All iPAQs are running OLSR routing protocol. The physical link will be broken in the middle of the test case to measure the routing protocol ability to reestablish the link.

Number	Name		
TC1.OLSR.4	4 Nodes, 3 hop in between.		
Environment, equipment and software used:		Client parameters:	
• Two 3850 and two 3900 iPAQs. • Familiar Linux. • RTP Client version 1.0. • Tcpdump. • UU OLSR version 0.4.5. 802.11b, 11Mbps, ch 10 (2GHz.)		Initial Jitter Buffer (msec) Payload length (GSM/RTP) Routing Protocol	60 3 OLSR
Topology			
			
Execution			
Start the client software. Speak for 2 min. Break the connection for 30 seconds. Reestablish the connection for another 2 mins.			

Plots

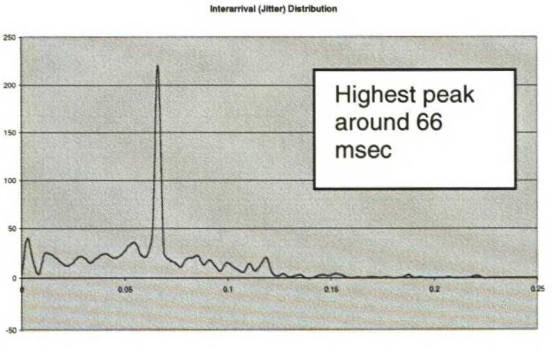
Inter-arrival (Jitter)



X-axis: time in seconds.

Y-axis: Inter-arrival amount in seconds.

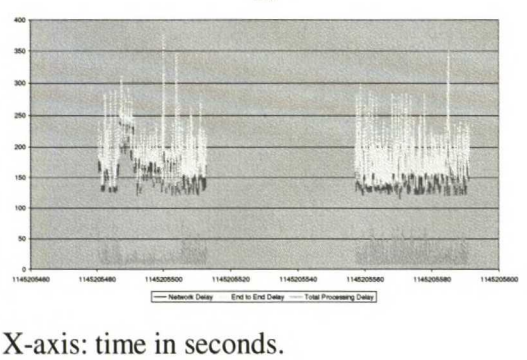
Inter-arrival (Jitter) Distribution



X-axis: Inter arrival time in seconds.

Y-Axis: Number of packets.

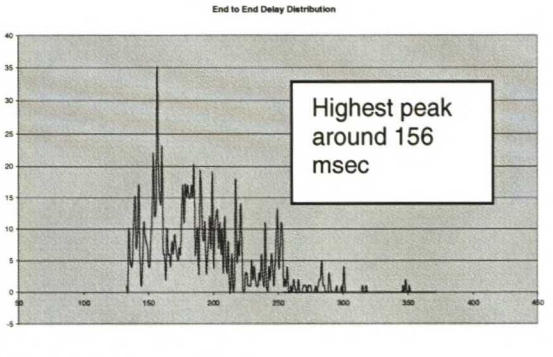
End to End Delay.



X-axis: time in seconds.

Y-axis: Delay in msec.

End to End Delay distribution.



X-axis: Time in msec.

Y-Axis: Number of packets.

Analysis

All values in msec	End to End Delay	Network Delay	Processing Delay	Inter-arrival time
Average	187	169	17	61
Std deviation	37	36	12	43
90% percentile	244	226	36	106

Packet loss (packet)	16
Re-establish time (second)	15

Protocol overhead (packets)	286 / 7969 = 3%	
Protocol overhead (bytes)	21220 / 1222673 = 1.73%	
	RTP (byte/packet)	OLSR (byte/packet)
Average packet size	153	74

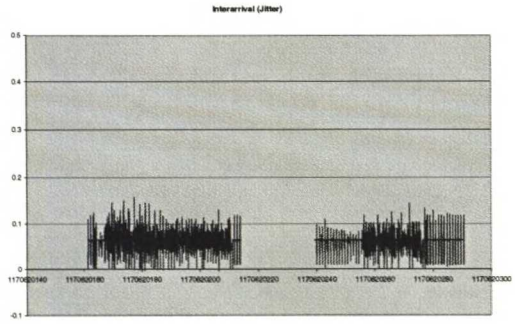
TC1.AODV.4

This test case is similar to TC1.OLSR.4, where 3 hops are involved with 4 iPAQs; two are acting as intermediate routers. All iPAQs are running AODV routing protocol. The physical link will be broken in the middle of the test case to measure the routing protocol ability to reestablish the link.

Number	Name		
TC1.AODV.4	4 Nodes, 3 hop in between.		
Environment, equipment and software used:		Client parameters:	
• Two 3850 and two 3900 iPAQs. • Familiar Linux. • RTP Client version 1.0. • Tcpdump. • UU AODV version 0.91. 802.11b, 11Mbps, ch 10 (2GHz)		Initial Jitter Buffer (msec)	60
		Payload length (GSM/RTP)	3
		Routing Protocol	AODV
Topology			
			
Execution			
Start the client software. Speak for 2 min. Break the connection for 30 seconds. Reestablish the connection for another 2 mins.			

Plots

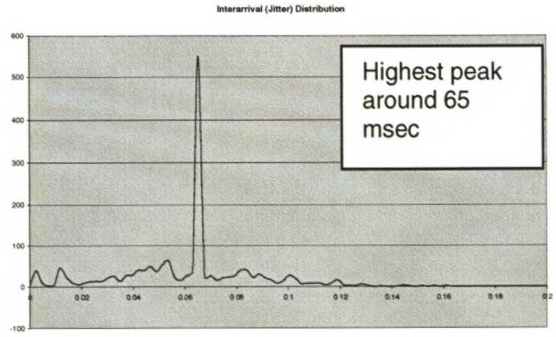
Inter-arrival (Jitter).



X-axis: time in seconds.

Y-axis: Inter-arrival amount in seconds.

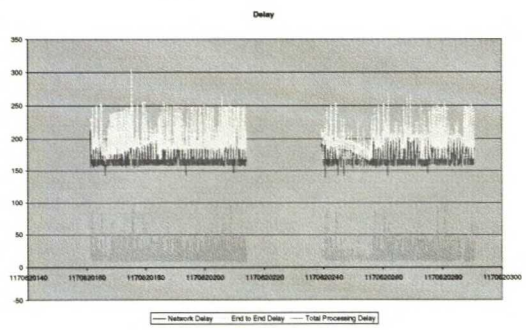
Inter-arrival (Jitter) Distribution.



X-axis: Inter arrival time in seconds.

Y-Axis: Number of packets.

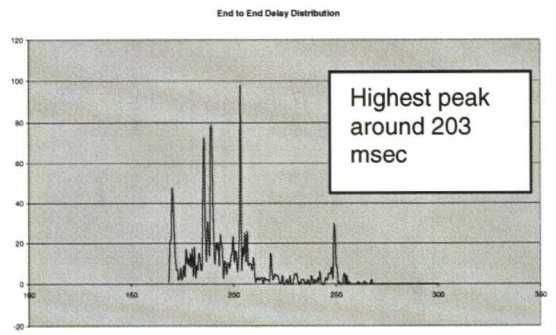
End to End Delay.



X-axis: time in seconds.

Y-axis: Delay in msec.

End to End Delay distribution.



X-axis: Time in msec.

Y-Axis: Number of packets.

Analysis

All values in msec	End to End Delay	Network Delay	Processing Delay	Inter-arrival time
Average	195	175	20	61
Std deviation	20	15	14	32
90% percentile	228	196	43	92

Packet loss (packet)	15	
Re-establish time (second)	1.5	
Protocol overhead (packets)	666 / 3665 = 18.17 %	
Protocol overhead (bytes)	46284 / 561207 = 8.24%	
	RTP (byte/packet)	AODV (byte/packet)
Average packet size	153	68

4.2.2. Test Case 1 Summary

(Values in msec) (Av = Average)	End to End Delay		Network Delay		Processing Delay		Inter-arrival time	Packet Loss	Re-est time	Protocol Overhead	
										Bytes	Packets
	Av	90%	Av	90%	Av	90%					
TC1.OLSR.2	158	187	141	171	16	38	60	3	0.5	1.57%	3.39%
TC1.OLSR.3	166	222	149	202	16	38	60	4	8	1.81%	3.86%
TC1.OLSR.4	187	244	169	226	17	36	61	16	15	1.73%	3.58%

(Values in msec) (Av = Average)	End to End Delay		Network Delay		Processing Delay		Inter-arrival time	Packet Loss	Re-est time	Protocol Overhead	
										Bytes	Packets
	Av	90%	Av	90%	Av	90%					
TC1.AODV.2	163	188	143	165	20	41	61	1	0.5	3.19%	7.22%
TC1.AODV.3	168	193	148	174	20	43	62	4	1	3.29%	7.38%
TC1.AODV.4	195	228	175	196	20	43	61	15	1.5	8.24%	18.17%

In test cases TC1.AODV.2 and TC1.OLSR.2, the end to end delay and network delay are very close because the routing protocol is only involved when the routing protocol is trying to discover the topology. The routing protocol frameworks only modify the operating system routing tables according to the topology. When a packet is routed to an intermediate node, the operating system performs packet forwarding based on its routing table. As a result, network delay is almost the same in OLSR and AODV test cases.

The end to end delay in OLSR case is 158 msec and in AODV case it is 163 msec. The network delay in OLSR case is 141.7 msec and in AODV case it is 143 msec. It is worth noting though

that AODV is consuming more CPU time than OLSR. In the case of OLSR, processing time is about 4 msec lower than AODV case.

As the number of hops increases, the end to end and network delays increase. The processing delay is almost constant at an average of 17.09 msec for OLSR cases and 20.58 msec for AODV cases. The packet loss also increases when the number of hops increases.

Figure 35 shows the end to end delay 90% percentile. When examining the 90% percentile, it becomes clear that even though the average end to end delay is lower in the cases of OLSR than in AODV cases, the 90% percentile is larger in the cases of OLSR. This implies that the end to end delay distribution is more spread in the case of OLSR. This can be explained by knowing that the OLSR protocol implementation performs topology update every 2 seconds. This will interfere in processing the RTP packets and will cause more RTP packets to be late, hence the wider delay distribution.

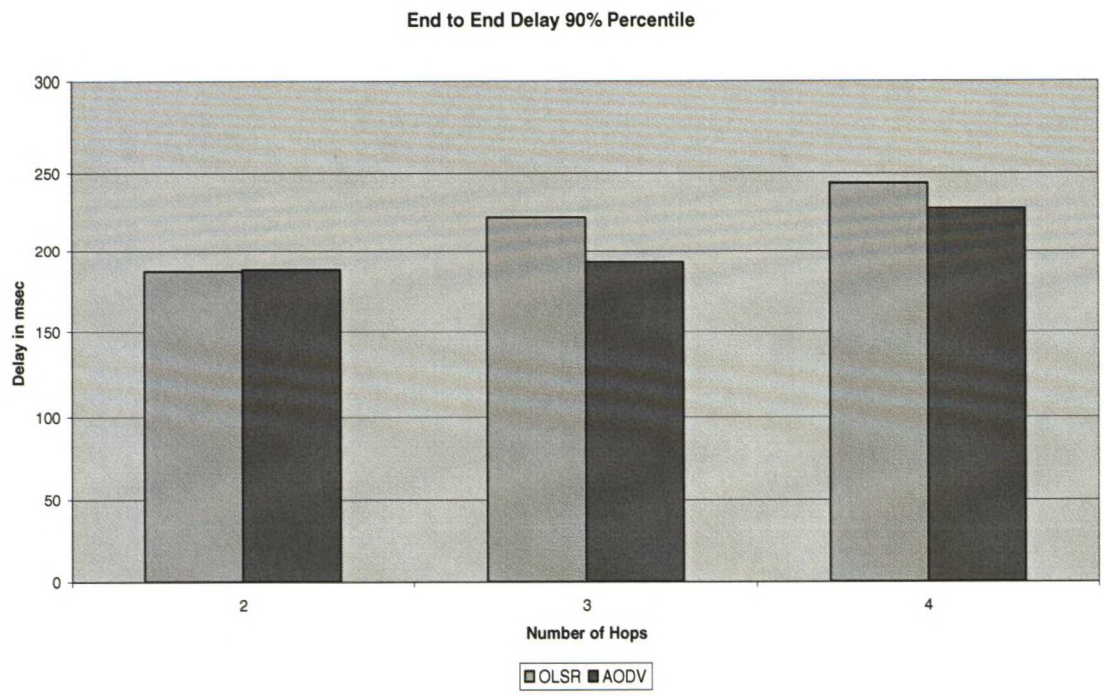


Figure 35 End to end delay 90% percentile

OLSR requires more time to re-discover the route after a broken link is re-established. This is an unexpected behavior since it is a proactive protocol, which performs periodic discovery of the current topology. Since the OLSR protocol implementation performs topology update every 2 seconds, it is expected to update the topology no later than 2 seconds. Nevertheless, this

result may indicate that the OLSR implementation used in this project has this deficiency. Figure 36 compares the delay between OLSR and AODV.

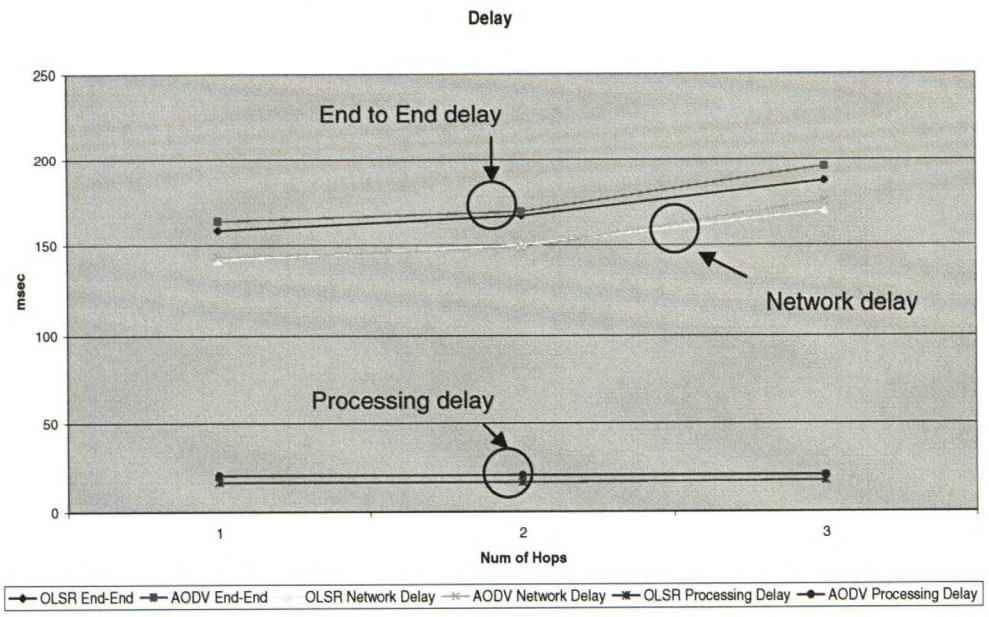


Figure 36 Delay comparison between OLSR and AODV

Figure 37 compares the protocol overhead of both OLSR and AODV.

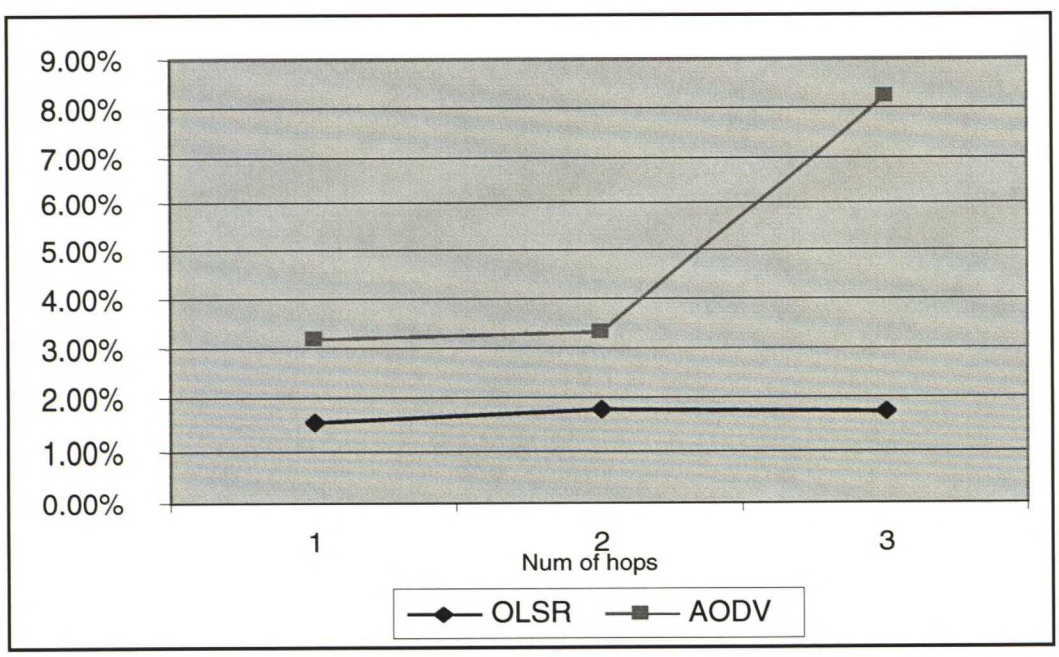


Figure 37 Protocol Overhead of OLSR and AODV

Although AODV is a reactive protocol and responds only when needed, it consumes a higher share of the bandwidth. As the number of hops increases, OLSR overhead slightly increases, whereas AODV overhead significantly increases. This can be explained by the fact that the test bed environment is not stable and the topology is updated tens of times during a test execution. Since AODV exchanges bigger packets, it follows that it will consume bigger share of the bandwidth if it is going to exchange many of them.

It is not possible however to judge the scalability of OLSR or AODV routing protocols using this project test bed. A scalability test can only be performed with a larger number of nodes. The observation in Figure 37 is that AODV is suited to a small size ad hoc network.

And finally, Figure 38 shows a comparison between reestablishing time for OLSR and AODV protocol.

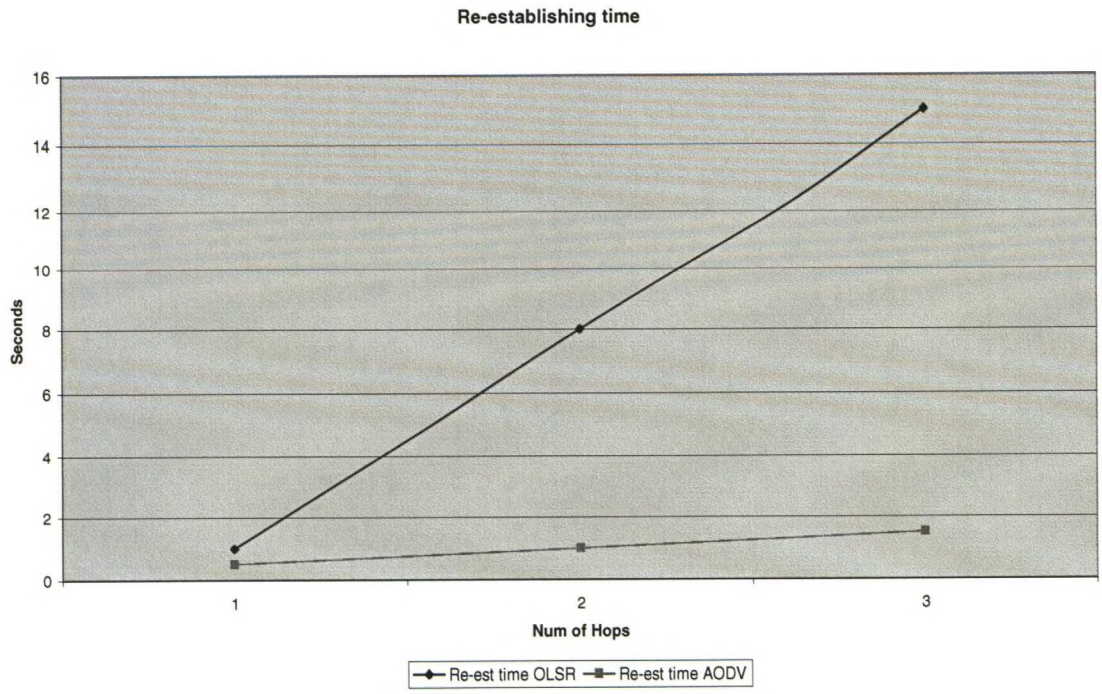


Figure 38 Reestablishing time for OLSR and AODV

4.3. Test Case 2: Effect of type of nodes

The objective of this test case is to measure the Qualify of Service factors as the type of nodes is changed. There are 2 classes of nodes: normal nodes and power nodes. All iPAQs are

classified as normal nodes, and all laptops are classified as power nodes. The classification is based on the processing power, memory size and battery life.

There are 2 test cases, each with 2 sub test cases, in all these cases, there are 3 hops and 4 nodes.

OLSR test cases:

- TC2.OLSR.Laptop
- TC2.OLSR.iPAQ (same as TC1.OLSR.4)

And AODV test cases:

- TC2.AODV.Laptop
- TC2.AODV.iPAQ (same as TC1.AODV.4)

The format of test cases names is TC2.<Protocol Name>.<Node type>. Finally, a summary section is included to present all the collected values in one place.

4.3.1. Test Case 2 sub test cases

TC2.OLSR.laptop

In this test case, there are 3 hops with 2 iPAQs and 2 laptops. The 2 laptops are acting as intermediate routing nodes. The iPAQs and laptops are running OLSR routing protocol. The physical link will be broken in the middle of the test case to measure the routing protocol ability to reestablish the link.

Number	Name		
TC2.OLSR.laptop	4 Nodes, 3 hop in between.		
Environment, equipment and software used:		Client parameters:	
• Two 3850 and two laptops. • Familiar Linux, fedora core Linux on laptops. • RTP Client version 1.0. • Tcpdump. • UU OLSR version 0.4.5. • 802.11b, 11Mbps, ch 10 (2GHz.)		Initial Jitter Buffer (msec)	60
		Payload length (GSM/RTP)	3
		Routing Protocol	OLSR

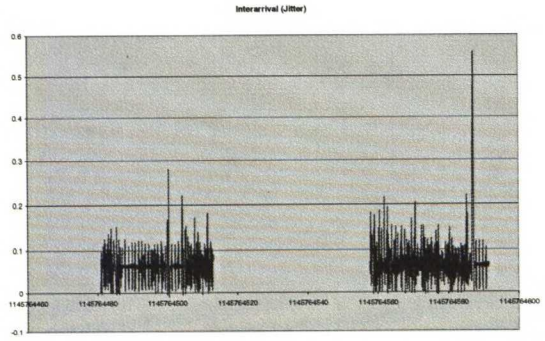
Topology:

Execution:

Start the client software. Speak for 2 min. Break the connection for 30 seconds. Reestablish the connection for another 2 mins.

Plots:

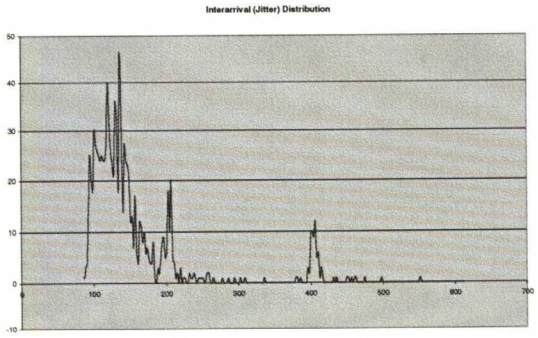
Inter-arrival (Jitter)



X-axis: time in seconds.

Y-axis: Inter-arrival amount in seconds.

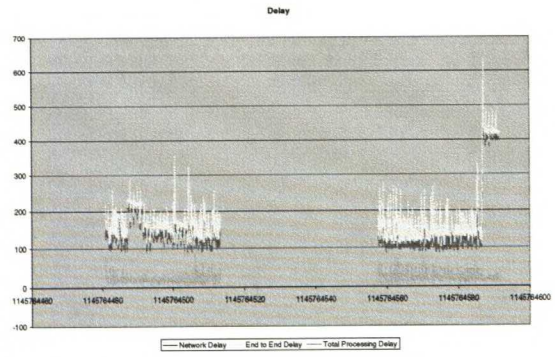
Inter-arrival (Jitter) Distribution



X-axis: Inter arrival time in msec.

Y-Axis: Number of packets.

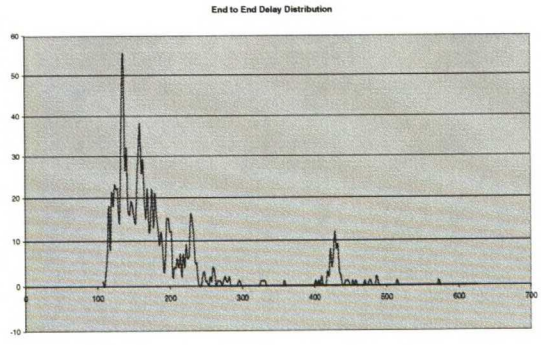
End to End Delay.



X-axis: time in seconds.

Y-axis: Delay in msec.

End to End Delay distribution.



X-axis: Time in msec.

Y-Axis: Number of packets.

Analysis:				
All values in msec	End to End Delay	Network Delay	Processing Delay	Inter-arrival time
Average	179	157	21	61
Std deviation	79	79	12	39
90% percentile	241	210	38	105

Packet loss (packet)	17	
Re-establish time (second)	15	
Protocol overhead (packets)	301 / 8360 = 3.60%	
Protocol overhead (bytes)	22274 / 1279080 = 1.74%	
	RTP (byte/packet)	OLSR (byte/packet)
Average packet size	153	74

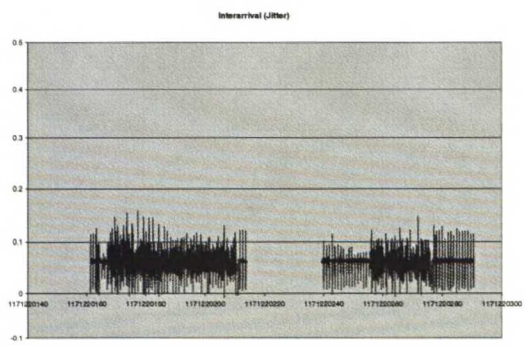
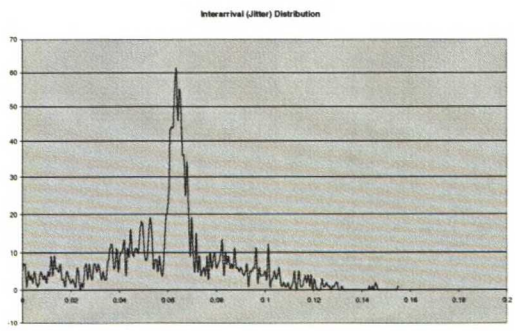
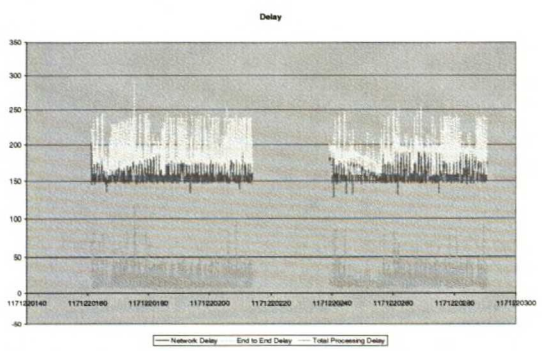
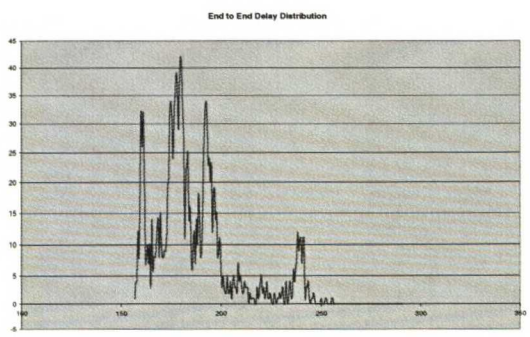
TC2.AODV.laptop

In this test case, there are 3 hops with 2 iPAQs and 2 laptops. The 2 laptops are acting as intermediate routing nodes. The iPAQs and laptops are running AODV routing protocol. The physical link will be broken in the middle of the test case to measure the routing protocol ability to reestablish the link.

Number	Name		
TC2.AODV.laptop	4 Nodes, 3 hop in between.		
Environment, equipment and software used:		Client parameters:	
• Two 3850 and two laptops. • Familiar Linux, fedora core Linux for laptops. • RTP Client version 1.0. • Tcpdump. • UU AODV version 0.91. • 802.11b, 11Mbps, ch 10 (2GHz.)		Initial Jitter Buffer (msec)	60
		Payload length (GSM/RTP)	3
		Routing Protocol	AODV

Topology

Execution
Start the client software. Speak for 2 min. Break the connection for 30 seconds. Reestablish the connection for another 2 mins.
Plots

Inter-arrival (Jitter)	Inter-arrival (Jitter) Distribution
	
X-axis: time in seconds. Y-axis: Inter-arrival amount in seconds.	X-axis: Inter arrival time in seconds. Y-Axis: Number of packets.
End to End Delay	**End to End Delay distribution**
	
X-axis: time in seconds. Y-axis: Delay in msec.	X-axis: Time in msec. Y-Axis: Number of packets.

Analysis				
All values in msec	End to End Delay	Network Delay	Processing Delay	Inter-arrival time
Average	185	165	20	60
Std deviation	20	15	14	26
90% percentile	218	186	43	92

Packet loss (packet)	13	
Re-establish time (second)	1.5	
Protocol overhead (packets)	712 / 3681 = 19.34 %	
Protocol overhead (bytes)	48424 / 563193 = 8.59%	
	RTP (byte/packet)	AODV (byte/packet)
Average packet size	153	68

4.3.2. Test Case 2 Summary

The following tables compare the results collected in test case 2 and those collected in TC1.OLSR.4 and TC1.AODV.4 for both OLSR and AODV respectively.

(Values in msec) (Av = Average)	End to End Delay		Network Delay		Processing Delay		Inter-arrival time	Packet Loss	Re-est time	Protocol Overhead	
	Av	90%	Av	90%	Av	90%				Bytes	Packets
TC2.OLSR.Laptop	179	241	157	210	21	38	61	17	15	1.74%	3.60%
TC1.OLSR.4	187	244	169	226	17	36	61	16	15	1.73%	3.58%

(Values in msec) (Av = Average)	End to End Delay		Network Delay		Processing Delay		Inter-arrival time	Packet Loss	Re-est time	Protocol Overhead	
										Bytes	Packets
	Av	90%	Av	90%	Av	90%					
TC2.AODV.Laptop	185	218	165	186	20	43	60	13	1.5	8.59%	19.34%
TC1.AODV.4	195	228	175	196	20	43	61	15	1.5	8.24%	18.17%

Having laptops with relatively higher processing capacity reduced the network delay, and eventually the end to end delay. The delay distribution is also more compressed in the cases of using intermediate laptops. It is an expected result because the laptops are faster in packet forwarding and routing protocols processing than the iPAQs. The time saved when using more powerful nodes in the middle is 8-10 msec. As a result, the iPAQs limited processing power limits the number of intermediate nodes in an ad hoc network, and hence the size of an ad hoc network.

4.4. Test Case 3: Topology change effect

The objective of this test case is to measure the quality of service factors when the topology is changed. Only iPAQs will be tested to eliminate the effect of using power nodes on the result of this test case.

There are 2 sub test cases:

- TC3.OLSR.
- TC3.AODV.

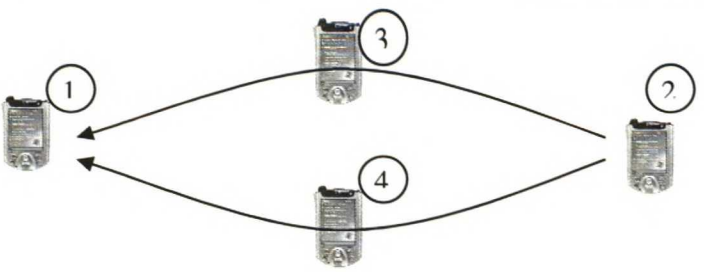
The format of the test cases names is TC3.<Protocol Name>. All 4 iPAQs will be used during this test case. Finally, a summary section is included to present all the collected values in one place.

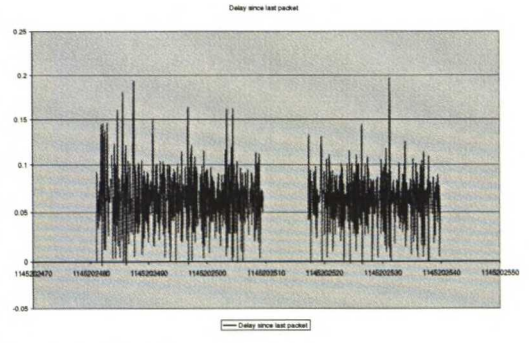
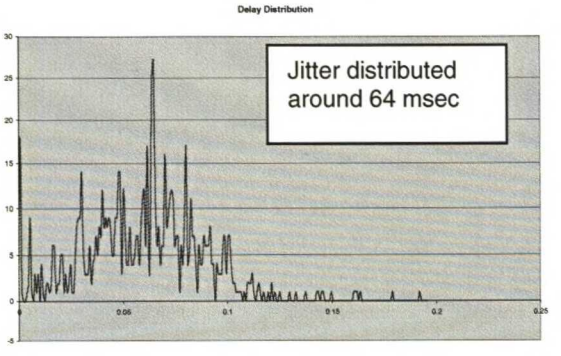
4.4.1. Test Case 3 sub test cases

TC3.OLSR

In this test case, there are 2 hops with 3 iPAQs. An alternative route will be created during the execution of the test case. This new route is made by adding a fourth iPAQ to the ad hoc

network. The fourth iPAQ is in the middle between the two end iPAQs. The topology change is then made by removing the original middle node from the ad hoc network. The routing protocol must discover the alternative route. All iPAQs are running OLSR routing protocol.

Number	Name		
TC3.OLSR	4 Nodes, 3 hop in between.		
Environment, equipment and software used:		Client parameters:	
• Two 3850 and 3900 iPAQs. • Familiar Linux. • RTP Client version 1.0. • Tcpdump. • UU OLSR version 0.4.5. • 802.11b, 11Mbps, ch 10 (2GHz.)		Initial Jitter Buffer (msec) Payload length (GSM/RTP) Routing Protocol	60 3 OLSR
Topology			
			
Execution			
Start the client software. Make sure that only node 3 exists as a middle routing node between node 1 and 2. After a while, bring node 4 in the ad hoc network and check that node 1 and 2 can now see node 4. Break the connection between node 1 and 3, and between node 2 and 3.			

Plots	
Inter-arrival (Jitter)	Inter-arrival (Jitter) Distribution
	
X-axis: time in seconds. Y-axis: Inter-arrival amount in seconds.	X-axis: Inter arrival time in seconds. Y-Axis: Number of packets.

Analysis				
All values in msec	End to End Delay	Network Delay	Processing Delay	Inter-arrival time
Average	121	99	22	60
Std deviation	21	22	14	29
90% percentile	142	123	41	95

Packet loss (packet)	132	
Re-establish time (second)	8	
Protocol overhead	144 / 4064 = 3.54%	
Protocol overhead (bytes)	10672 / 624886 = 1.71%	
	RTP (byte/packet)	OLSR (byte/packet)
Average packet size	153	74

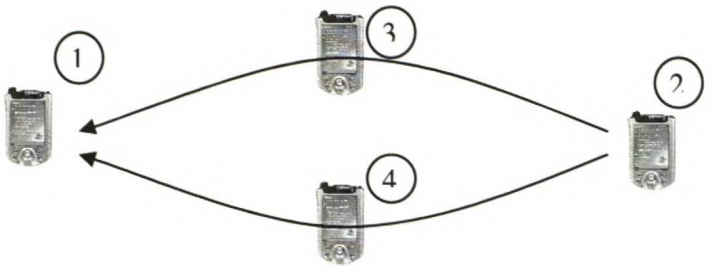
TC3.AODV

In this test case, there are 2 hops with 3 iPAQs. An alternative route will be created during the execution of the test case. This new route is made by adding a fourth iPAQ to the ad hoc network. The fourth iPAQ is in the middle between the two end iPAQs. The topology change is then made by removing the original intermediate node from the ad hoc network. The routing protocol must discover the alternative route. All iPAQs are running AODV routing protocol.

Number	Name
TC3.AODV	4 Nodes, 3 hop in between.

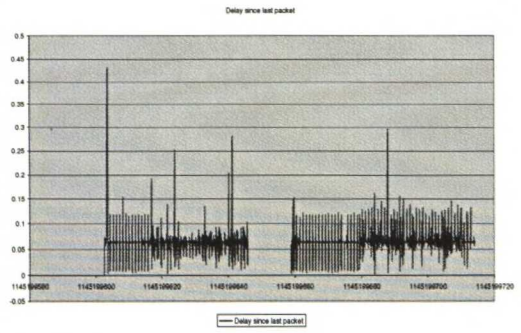
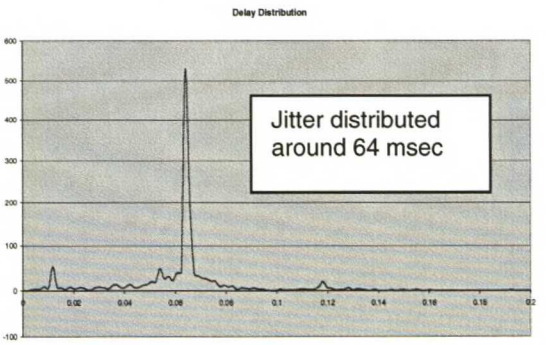
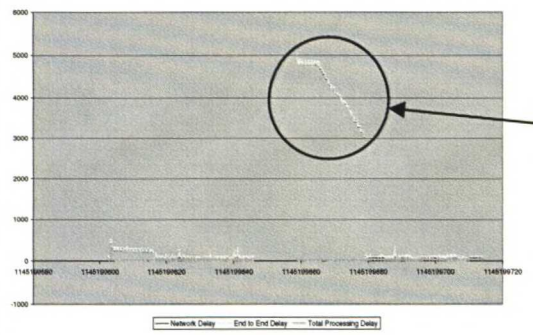
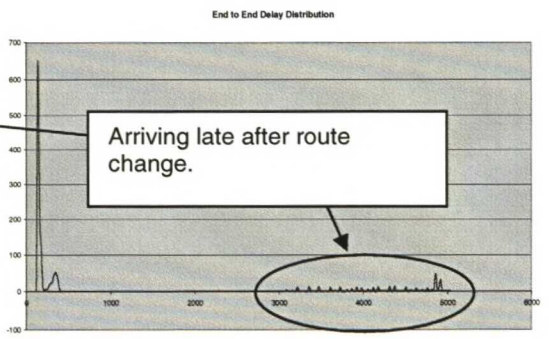
Environment, equipment and software used:	Client parameters:	
• Two 3850 and two 3900 iPAQs. • Familiar Linux. • RTP Client version 1.0. • Tcpdump. • UU AODV version 0.91. • 802.11b, 11Mbps, ch 10 (2GHz.)	Initial Jitter Buffer (msec)	60
	Payload length (GSM/RTP)	3
	Routing Protocol	AODV

Topology



Execution

Start the client software. Make sure that only Node 3 exists as a middle routing node between node 1 and 2. After a while, bring in node 4 and check that node 1 and 2 can now see node 4. Break the connection between node 1 and 3, and between node 2 and 3.

Plots	
Inter-arrival (Jitter)  X-axis: time in seconds. Y-axis: Inter-arrival amount in seconds.	Inter-arrival (Jitter) Distribution  X-axis: Inter arrival time in seconds. Y-Axis: Number of packets.
End to End Delay  X-axis: time in seconds. Y-axis: Delay in msec.	End to End Delay distribution  X-axis: Time in msec. Y-Axis: Number of packets.

Analysis				
All values in msec	End to End Delay	Network Delay	Processing Delay	Inter-arrival time
Average	1022	1006	16	63
Std deviation	1715	1718	15	26
90% percentile	4450	4439	36	78

Packet loss (packet)	178
Re-establish time (second)	7

Protocol overhead	526 / 6786 = 7.75%	
Protocol overhead (bytes)	35836 / 1040022 = 3.45%	
	RTP (byte/packet)	AODV (byte/packet)
Average packet size	153	68

Figure 39 shows a typical real time management performed by the VoIP software client. The time management shown in Figure 39 is obtained during executing test case TC3.AODV

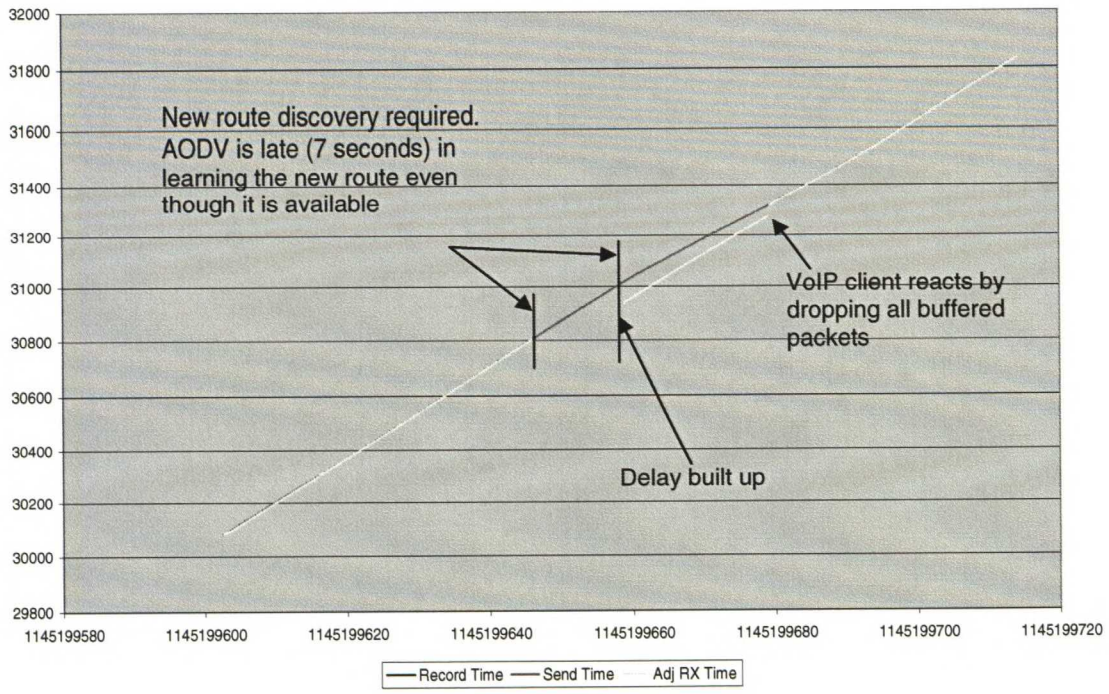


Figure 39 Time management example

Figure 39 shows the increasing end to end delay due to late route re-discovery by AODV. The audio becomes 5 seconds late. The VoIP client decides to drop all buffered packets to reduce delay time to normal. After which, audio arrives after normal delay.

4.4.2. Test Case 3 Summary

	End to End Delay	Network Delay	Processing Delay	Inter-arrival time	Packet Loss	Re-est time	Protocol Overhead
TC3.OLSR	121	99	22	60	132	8	3.54%
TC3.AODV	1022	1006	16	63	178	7	7.75%

It is observed that OLSR requires more processing power but takes less share of the bandwidth, almost half of that needed for AODV. Discovering a new route takes about the same time from both protocols. However, when audio is resumed in OLSR case, the end to end delay is normal, but in AODV case, the end to end delay is very high. In almost all trials of AODV test case, the VoIP software client had to drop the buffer in order to keep the end to end delay low.

This test case clearly shows the advantage of OLSR over AODV in topology changes scenarios. However, due to limited hardware availability, it was not possible to run other scenarios. A bigger network topology might produce different results.

Chapter 5

5. Conclusion

After executing many test cases on the test bed, there are a number of conclusions that can be drawn:

1. The end to end delay in OLSR is lower than in AODV. This is expected since OLSR is a proactive routing protocol.
2. The processing overhead of AODV is higher than OLSR. This is unexpected since AODV is reactive.
3. AODV demands more processing power from the end iPAQs. But when discovering an alternative route, OLSR consumes more processing power.
4. OLSR is slower in re-establishing a broken link. This is an unexpected result.
5. AODV has a higher share of the bandwidth. This is another unexpected result that may affect the scalability of AODV routing protocol.
6. AODV is more suited for small size ad hoc network that may require fast route re-establishment after link break.
7. Using powerful middle routing nodes will reduce network delay. This will allow for a larger ad hoc network with the same level of quality of service.
8. OLSR is better in recovering from route changes during a call. It can be attributed to the limited processing power of the iPAQs that the AODV introduces a long period of delay in the end to end delay.

Although the results obtained depend on the routing protocols implementation, each protocol serves part of the real-time nature of VoIP calls, but neither of them completely fulfills all the requirements. AODV requires greater bandwidth and processing overhead share but results in lower re-establishing time, while OSLR requires less bandwidth and processing overhead but results in higher re-establishing time.

It worth mentioning that in experimental analysis, there is a larger number of variables, such as the routing protocol specific implementation. This makes drawing a clear conclusion much harder. It was not possible to perform deeper analysis of the reasons behind the obtained results by collecting more information from the iPAQs because of the limited processing power of the iPAQs.

Chapter 6

6. Future work

It would be interesting to conduct more experiments involving newer routing frameworks, perhaps a hybrid routing protocol between OLSR and AODV that combines the advantages of both.

The VoIP software client can also be improved to incorporate newer more adaptive audio or speech codecs. Newer techniques of real time audio management could be investigated as well. Further development can be conducted to include conference calls, broadcasting and video calls.

During this project, SIP as a signaling protocol in an ad hoc network is not very useful, especially if IP addresses are the only mean of addressing nodes. SIP in ad hoc environments can be investigated, such as distributed SIP infrastructure.

7. References

1. Programmers Guide to OSS http://www.opensound.com/pguide/index.html
2. Ethereal user guide: http://www.ethereal.com/docs/user-guide/
3. ITU-T recommendation: Guidance on one-way delay for Voice over IP http://www.itu.int/rec/recommendation.asp?type=items&lang=E&parent=T-REC-G.114-200309-P!AppII
4. J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler: "SIP: Session Initiation Protocol," Request for Comments 3261, Internet Engineering Task Force, June 2002. http://www.ietf.org/rfc/rfc3261.txt
5. H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," Request for Comments 1889, Internet Engineering Task Force, Jan. 1996. http://www.ietf.org/rfc/rfc1889.txt
6. H.323 : Packet-based multimedia communications systems http://www.itu.int/rec/T-REC-H.323/
7. SIP Introduction, Jan Janak, Copyright © 2003 FhG FOKUS, A brief overview of SIP describing all important aspects of the Session Initiation Protocol. http://www.iptel.org/ser/doc/sip_intro/sip_introduction.html
8. M. Handley, V. Jacobson, "SDP: Session Description Protocol", April 1998, Request for Comments 2327, Internet Engineering Task Force, http://www.ietf.org/rfc/rfc2327.txt
9. Ad hoc routing protocol list, From Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Ad_hoc_routing_protocol_list
10. Ad hoc routing protocol list, From Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Ad_hoc_routing_protocol_list
11. Route assignment, From Wikipedia, the free encyclopedia (Redirected from Routing protocol), http://en.wikipedia.org/wiki/Routing_protocol
12. Ad-hoc On-demand Distance Vector, From Wikipedia, the free encyclopedia, (Redirected from AODV), http://en.wikipedia.org/wiki/AODV
13. S-38.188 Computer networking, Spring 2004 Lecture 4 , http://www.netlab.tkk.fi/opetus/s38188/2004/eindex.shtml

14. Optimized link state routing protocol, From Wikipedia, the free encyclopedia, (Redirected from OLSR), http://en.wikipedia.org/wiki/OLSR
15. Codec Feature Comparison, SPEEX.ORG, http://www.speex.org/comparison.html
16. Voice Over IP - Per Call Bandwidth Consumption, Cisco Systems, http://www.cisco.com/en/US/tech/tk652/tk698/technologies_tech_note09186a0080094ae2.shtml
17. GSM 06.10 lossy speech compression, http://kbs.cs.tu-berlin.de/~jutta/toast.html
18. IP Telephony: Packet-based multimedia communications systems, Olivier Hersent, David Gurle, Jean-Pierre Petit. P.78.
19. UMTS Bearer Service Attributes, Nokia, http://www.3gpp.org/ftp/tsg_sa/WG2_Arch/TSGS2_04/tempdoc/S2-99225.doc
20. JRTPLIB v2.x, Jori Liesenborgs, jori@lumumba.uhasselt.be, http://research.edm.luc.ac.be/jori/jrtplib/jrtplib_old.html
21. Adam Roach, Ericsson Inc., Ringback tones in SIP-Based Telephony, Internet Draft, Internet Engineering Task Force, November 2000, http://www.cs.columbia.edu/sip/drafts/draft-roach-voip-ringtone-00.txt
22. Signals and Circuit Conditions of USA and Canadian Telephone Systems, http://www.telephonetribute.com/signal_and_circuit_conditions.htm
23. Differentiated services, From Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Diffserv
24. Integrated services, From Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Intserv
25. Resource Reservation Protocol, From Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Resource_Reservation_Protocol
26. IEEE 802.11e, From Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/IEEE_802.11e
27. GenDet Call Progress Detector Performance, Miller Engineering Services, Inc, http://www.mesi.net/MESiWeb/perform.htm

8. APPENDIX

8.1. Compiling

8.1.1. RTP

RTP client consists of the following code files:

- mediasession.cpp
- mediasession.h
- rtpbuffer.cpp
- rtpbuffer.h
- audiotesting.cpp

Audiotesting is a simple starter application that uses the RTP client module (mediasession class). To compile the binary used in testing, you should compile everything including audiotesting.cpp, this will generate a stand alone application. Use "intelcompile" shell script to compile the application for i386 platform, and "armcompile" to compile for ARM platform. You can open these shell scripts and inspect them, they are simple g++ command, and here are the contents of these scripts:

intelcompile.sh

```
g++ -o voip audiotesting.cpp mediasession.cpp rtpbuffer.cpp
-I/usr/local/include/gsm -I/usr/local/include/jrtplib -L/usr/local/lib -ljrtp
-lgsm -lpthread -Wl,-R/usr/local/lib -Wno-deprecated
```

And armcompile.sh

```
arm-Linux-g++ -o armat audiotesting.cpp mediasession.cpp rtpbuffer.cpp
-I/usr/local/include/gsm -I/usr/local/include/jrtplib -L/usr/local/lib/arm
-ljrtp -lgsm -lpthread -Wl,-R/usr/local/lib/arm -Wno-deprecated
```

8.1.2. JRTP Library

This project uses JRTPLib version 2.9, you can get it here: http://research.edm.luc.ac.be/jori/jrtplib/jrtplib_old.html. Note that the code is not compatible with version 3.0 as the author has introduced major changes that broke compatibility.

Compile and install the library as instructed to get a working RTP stack on i386 platform. To compile for ARM, configure for ARM processor then run make.

```
configure --host=arm
make
```

The output should be in .lib subdirectory, which is a shared object named libjrtp.so.2.9.0.

8.1.3. GSM codec

Download the GSM codec from http://kbs.cs.tu-berlin.de/~jutta/toast.html then configure, run make, and make install to get a working binary for i386. To compile for ARM:

```
configure --host=arm
make
```

Note that a cross compiler must be install if you want to compile for ARM platform

8.1.4. Copying

Now that you have SIP, RTP, JRTP and GSM codec compiled, you can simply copy them to the iPAQ and start using them. Copy JRTP (libjrtp.so.2.9) and GSM codec (libgsm.so.1.0.10) shared objects to /usr/lib on the iPAQ, you may want to create a symbolic link to these files if you run into problems, then copy SIP and armat to your home directory.

8.2. User guide

The VoIP client consists of the following files:

- The binary file, named SIP.
- 1.cfg, which is a text file containing the required configurations

The configuration file consists of the following items. Each item is placed individually in a separate line:

```
SIPLocalPort
SIPRemotePort
RTPLocalPort
RTPRemotePort
RTPFlags
TransmissionPeriod
SamplingFreq
RingBackRingTime
RingBackSilentTime
BusyRingTime
BusySilentTime
AudioInputFile
AudioOutputFile
LocalIPAddress
RingSignalFile
```

```
RingToneFile
FriendlyName
```

As an example:

```
5060
5060
55000
55000
45
20
8000
1000
3000
500
500
in.gsm
out.gsm
10.0.0.5
ring3.wav
nokiatune.wav
a1
```

To start the SIP client, simply run the binary and pass the configuration file name as the only parameter:

```
./sip 1.cfg
```

The first thing displayed are the parameters the VoIP client is using. This acts as confirmation to the user about the used parameters values. Note that there are many customizable parameter values to allow for easier debugging.

```
############## REGISTRY ############
Local SIP Port : 5060
Remote SIP Port: 5080
Local RTP Port : 55000
Remote RTP Port: 44000
RTP Flags      : 45
Sampling Freq  : 8000
Audio in file  : in.gsm
Audio out file : out.gsm
Local IP Addr  : 127.0.0.1
####################################
ReceiveThread started...
```

```
ProcessThread started...
SIP>
```

The last thing the user will get is a command prompt. If you enter "help" you will get a list of possible commands. This appendix section also describes these commands.

```
SIP> help
Available commands:
  exit
  invite
  message
  help
  session
  sessions
SIP>
```

exit: exits the VoIP client.

invite: sends out a SIP INVITE message, i.e. initiate a new call; example: invite <ip address>

message: sends a SIP MESSAGE; as an example: message <ip address> <message body>

session: A command that helps executing other commands on a specific session, possible sub commands are:

session <session index> ok: to accept the incoming call

session <session index> cancel: to cancel/reject incoming call

session <session index> bye: to end the current active call.

session index is the index of the related session, it can be retrieved by the sessions command.

sessions: list all active sessions

Making a call

To make call, issue an invite command, like this:

```
SIP> invite 127.0.0.1

---------- SENDING REPORT ----------
Sending to 127.0.0.1:5080...
_____________________________________SIP
INVITE sip:localhost.localdomain SIP/2.0
Via: SIP/2.0/UDP 127.0.0.1
To: sip:127.0.0.1
From: Netlab1 <sip:localhost.localdomain@127.0.0.1>;tag=06106000012
```

```
Call-ID: 221151460
CSeq: 1 INVITE
Contact: Netlab1 <sip:localhost.localdomain@127.0.0.1>
Content-Type: plain/text
Content-Length: 29

tp=00
sf=00000
af=00
lp=50000
_____________________________________SIP
>> 327 byte(s) sent <<
```

The receiver will get the following notification:

```
---------- RECEIVE REPORT ----------
_____________________________________SIP

INVITE sip:localhost.localdomain SIP/2.0
Via: SIP/2.0/UDP 127.0.0.1
To: sip:127.0.0.1
From: Netlab1 <sip:localhost.localdomain@127.0.0.1>;tag=06106000012
Call-ID: 221151460
CSeq: 1 INVITE
Contact: Netlab1 <sip:localhost.localdomain@127.0.0.1>
Content-Type: plain/text
Content-Length: 29

tp=00
sf=00000
af=00
lp=50000
_____________________________________SIP
```

And immediately sends out a 180 RINGING SIP message:

```
---------- SENDING REPORT ----------
Sending to 127.0.0.1:5060...
_____________________________________SIP
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP 127.0.0.1:5080
To: <sip:127.0.0.1:5080>;tag=06106000012
From: Netlab1 <sip:localhost.localdomain@127.0.0.1:5080>;tag=06106000012
Call-ID: 221151460
CSeq: 1 INVITE
Contact: Netlab1 <sip:localhost.localdomain@127.0.0.1:5080>
Content-Type: plain/text
```

```
Content-Length: 28

tp=00
sf=00000
af=00
lp=5000
_____________________________________SIP
```

The sender will get the 180 RINGING response and display it:

```
---------- RECEIVE REPORT ----------
_____________________________________SIP

SIP/2.0 180 Ringing
Via: SIP/2.0/UDP 127.0.0.1:5080
To: <sip:127.0.0.1:5080>;tag=06106000012
From: Netlab1 <sip:localhost.localdomain@127.0.0.1:5080>;tag=06106000012
Call-ID: 221151460
CSeq: 1 INVITE
Contact: Netlab1 <sip:localhost.localdomain@127.0.0.1:5080>
Content-Type: plain/text
Content-Length: 28

tp=00
sf=00000
af=00
lp=5000
_____________________________________SIP
```

The sender will also sound the status ring back tone, and the receiver will sound the notification ringing tone. Both ends will now activate their audio components, and the following will be displayed as information:

```
NewSessionI: -1
SessionID: -1
######## AUDIO DEVICE REPORT ########
        Setting audio format
        Setting mono
        Setting sampling speed to 8000
        Setting full duplexing
        Audio fragment size: 1024 bytes
        Input buffer total fragments: 0
        Input buffer fragment size: 1024
        Output buffer total fragments: 4
        Output buffer fragment size: 1024
####################################
```

```
RingerPeriod = 1000, Sleep: 3000
```

The receiver accepts the call by issuing the following command:

```
SIP> session 0 ok
---------- SENDING REPORT ----------
Sending to 127.0.0.1:5060...
_____________________________________SIP
SIP/2.0 200 OK
Via: SIP/2.0/UDP 127.0.0.1:5080
To: <sip:127.0.0.1:5080>;tag=06106000012
From: Netlab1 <sip:localhost.localdomain@127.0.0.1:5080>;tag=06106000012
Call-ID: 221151460
CSeq: 1 INVITE
Contact: Netlab1 <sip:localhost.localdomain@127.0.0.1:5080>
Content-Type: plain/text
Content-Length: 28

tp=00
sf=00000
af=00
lp=5000
_____________________________________SIP
```

Which the sender will receive and will be displayed as:

```
---------- RECEIVE REPORT ----------
_____________________________________SIP


SIP/2.0 200 OK
Via: SIP/2.0/UDP 127.0.0.1:5080
To: <sip:127.0.0.1:5080>;tag=06106000012
From: Netlab1 <sip:localhost.localdomain@127.0.0.1:5080>;tag=06106000012
Call-ID: 221151460
CSeq: 1 INVITE
Contact: Netlab1 <sip:localhost.localdomain@127.0.0.1:5080>
Content-Type: plain/text
Content-Length: 28

tp=00
sf=00000
af=00
lp=5000
_____________________________________SIP
```

Then the sender replies back with an ACK message:

```
---------- SENDING REPORT ----------
Sending to 127.0.0.1:5080...
_____________________________________SIP
ACK sip:localhost.localdomain SIP/2.0
Via: SIP/2.0/UDP 127.0.0.1
To: <sip:127.0.0.1>;tag=06106000012
From: Netlab1 <sip:localhost.localdomain@127.0.0.1>;tag=06106000012
Call-ID: 221151460
CSeq: 1 INVITE
Contact: Netlab1 <sip:localhost.localdomain@127.0.0.1>
Content-Type: plain/text
Content-Length: 29

tp=00
sf=00000
af=00
lp=50000
_____________________________________SIP
```

And finally, the receiver will receive the ACK and display it:

```
---------- RECEIVE REPORT ----------
_____________________________________SIP

ACK sip:localhost.localdomain SIP/2.0
Via: SIP/2.0/UDP 127.0.0.1
To: <sip:127.0.0.1>;tag=06106000012
From: Netlab1 <sip:localhost.localdomain@127.0.0.1>;tag=06106000012
Call-ID: 221151460
CSeq: 1 INVITE
Contact: Netlab1 <sip:localhost.localdomain@127.0.0.1>
Content-Type: plain/text
Content-Length: 29

tp=00
sf=00000
af=00
lp=50000
_____________________________________SIP
```

Now, both parts will start RTP, and the following will be displayed:

```
stop all ringers
Starting RTP
```

When accepting a call, the command is "session 0 ok". The index "0" is the session index. If the command "sessions" is entered, all active sessions will be displayed, on the sender part

Index	Code	Call ID	From Tag	To Tag	Remote URL
=====	====	=======	========	======	==========
0	INVSENT	221151460	06106000012	-	127.0.0.1
1	-	-	-	-	-
2	-	-	-	-	-
3	-	-	-	-	-
4	-	-	-	-	-
5	-	-	-	-	-
6	-	-	-	-	-
7	-	-	-	-	-
8	-	-	-	-	-
9	-	-	-	-	-

And on the receiver part:

Index	Code	Call ID	From Tag	To Tag	Remote URL
=====	====	=======	========	======	==========
0	INVRCVD	221151460	06106000012	06106000012	127.0.0.1
1	-	-	-	-	-
2	-	-	-	-	-
3	-	-	-	-	-
4	-	-	-	-	-
5	-	-	-	-	-
6	-	-	-	-	-
7	-	-	-	-	-
8	-	-	-	-	-
9	-	-	-	-	-

Note that the Code refers to the state the session is in. For more information, refer to the state machine design in section 3.2.3.

At this point, RTP is active and the conversation can begin. Ending the call is done by this command: session 0 bye. To end the VoIP client, just type exit.

RTP testing

To facilitate RTP testing, automated testing scripts and an excel sheet with macros are created. The following are the scripts and their use:

fw.sh: this script helps setting up iptables to block or unblock nodes based on their MAC address. The MAC address of all the iPAQs and laptops are added in the script file. The script usage is like:

```
./fw.sh [i2|i4|i5|i6|l1|l2] [b|u]
```

i2, i4, i5, i6, l1, and l2 are the node identifier. b means block and u means unblock

start.sh: a pre-test script, it will clean up old collected data, sets the date, starts tcpdump and starts the time difference collector. All these operations are necessary before every test case.

setdate.sh: a script that will set the date to a specified date and time. It is important during the testing to synchronize the times on the iPAQs so as to collect consistent logs.

```
./setdate.sh mmddHHMMyyyy
```

tsc.sh: a script that will keep running tsa binary and append the result to a log file, whose name is specified as a parameter:

```
./tsc.sh <log file name>
```

sc.sh: this script is the upper level wrapper to all the smaller scripts. It helps running a scenario by taking a set of parameters.

```
./sc.sh datetime other_node conn_per_1 disconn_per_2 conn_perd_3 and rest of RTP
```

The date time parameter will be directly fed to setdate.sh, other_node will be used to block/unblock using fw.sh during the execution of the test scenario, conn_per_1, disconn_per_2 and conn_perd_3 are the period of times for the initial connection time, disconnection period time and the last period of connection.

exec.sh: it simply runs the RTP client.

finish.sh: it is run immediately after the end of the scenario. It stops the RTP, ends tcpdump and stop clock difference collection.

run.sh. This is the command that actually run the whole test case. It is an outer interface for the whole testing framework. It is used like this:

```
./run.sh date testcase_config
```

Where date is date and time in this format: mmddHHMMyyyy, as an example, 20:15, 14th April 2006 will be: 041420152006.

An example testcase_config,

```
i6 30 10 30 10.0.0.6 55000 55000 45 3 10 in.wav out.wav AODV
```

where the fields are as following:

i6: is the other calling end. It is necessary because the local calling nodes needs to know which remote calling node to block using fw.sh.

30: first connection duration, for which the connection will be kept on, it is in seconds.

10: once the first connection time is over, the link is broken using fw.sh script. The disconnection time is kept for the duration specified by this parameter, it is in seconds.

30: the second connection time after the disconnection period. The link is enabled by reconfiguring the firewalls using fw.sh. It is in seconds.

10.0.0.6: the other (remote) end IP address.

55000: the RTP port of the local end.

55000: (second one) is the RTP port of the remote end.

45: Is the RTP component flags, they are a bit mask as following:

1: Enable receiver.

2: Write everything received to a file.

4: Play out everything received to the speaker.

8: Enable transmitter.

16: Read audio from a file.

32: Read audio from the microphone.

Therefore, 45=1+4+8+32=Enable both receiver and transmitter, and use the microphone to read and the speakers to play out.

3: The number of GSM audio frames in every RTP packet. This number can be changed at runtime by simply typing the number and then pressing enter.

10: are the maximum allowable GSM audio frames in each RTP packet.

in.wav: in case of using files to read audio data, this is the file name.

out.wav: in case of using files to write audio data, this is the file name.

AODV|OLSR: this specifies which kind of test case this is going to be. It must be mentioned though that it is not used but it is kept if in the future a need for it arises again.

Once the testing is complete, a set of dat files are generated, they are:

dump.dat: the tcpdump, can be opened and analyzed using ethereal

tx_timing.dat: the transmitter logs

rx_timing.dat and pb_timing.dat: the receiver logs

td.dat: the clock snapshots collected by tcs.sh

Use addnewrun.sh script to load these files to desktop or laptop, the script works as following:

```
./addnewrun.sh <run num> <ip of 1st node> <ip of 2nd node>
```

Upload these files from both nodes participating in call to a windows machine with excel. Open the template excel sheet and run Tools>Macros>Run Macro...>DoMagic macro. You will be asked about the location of the logs, provide the location created by addnewrun.sh script. You

will then be asked about the direction of traffic to do the analysis. Once the macro is done, you should have a complete analysis of the logs in plots.